Universidad
Carlos III de Madrid
www.uc3m.es

TRABAJO FIN DE GRADO

Título: MODELADO 3D DE OBJETOS A PARTIR DE LA KINECT

Autor: Juan Toribios Blázquez

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Tutor: Luis Enrique Moreno Lorente

Cotutor: Fernando Martín Monar

Fecha: Septiembre 2012

Resumen

Este trabajo consiste en el desarrollo de un software para la captura de imágenes con la cámara de visión 3D Kinect y tras la realización de un análisis determinar figuras geométricas. Un robot, el cual se desenvuelve entre personas, debe interactuar con los objetos que le rodean y ser capaz de reconocerlos. De ahí nace la necesidad del modelado 3D de objetos. En este proyecto se trabaja con el algoritmo RANSAC para el reconocimiento de modelos geométricos y con el algoritmo k-d Tree, a través de librerías PCL en el lenguaje C++.

A partir de nubes de puntos captadas con la cámara RGB y el sensor de infrarrojos de la cámara Kinect, se tratan las imágenes para obtener un objeto sobre una superficie plana. Con el algoritmo k-d Tree se acelera la búsqueda de vecinos para determinar las normales del objeto, característica necesaria para el reconocimiento de formas, que junto a los modelos geométricos y el algoritmo RANSAC, permite determinar prismas rectangulares, esferas y cilindros. El software es probado con imágenes en las que se encuentran objetos cotidianos como pelotas, cajas, latas, etc. Se determina la forma geométrica que los caracteriza y el coste computacional de la búsqueda.

Abstract

This project starts with the development of a software program in order to capture images with the 3D vision camera (Kinect). An analysis to identify geometric figures is also carried out. A robot that interacts with people must interact with those objects surrounding it and be able to recognize them. The need tomodel 3D objects was born out of that reason.The RANSAC algorithm for the identification of geometric models and the k-d Tree algorithm have been used throughout the development of this project through PCL libraries in the C++ language.

Pictures are manipulated from points clouds captured with the RGB camera and the infrared sensor of the Kinect camera in order to get an object on a flat surface. The k-d trees accelerate the search of the nearest neighbours, which is a necessary feature when determining the object´s normal. This feature is needed for the identification of shapes which along with geometric models and the RANSAC algorithm allows to identify rectangular prisms, spheres and cylinders. The software is tested with pictures containing everyday objects such as balls, boxes, cans, etc. The geometric shape that characterizes these objects and the computational cost of the search are determined.

Índice general

Índice de figuras

1. Introducción

En la actualidad, el mundo de la robótica evoluciona rápidamente adaptándose a los constantes avances en la ciencia y en la sociedad. Una industria cada día mas flexible necesita una automatización estable. En el mundo de la salud la robótica tiene que dar respuesta a estas demandas que se generan debido al envejecimiento de la población, las personas con discapacidades y las operaciones a distancia. En entornos de peligro como excavaciones, desastres naturales y conflictos de guerra, los robots dan un apoyo en los momentos que se puede evitar poner en peligro una vida humana. En la Figura1.1 el humanoide RH1 desarrollado en el laboratorio de robótica de la UC3M [1].

Figura 1.1: Humanoide RH1 en la UC3M

Según la ISO un robot es: “Un manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular cargas, piezas, herramientas o dispositivos especiales, según trayectorias programadas para realizar tareas diversas”. Para que un robot trabaje y se relacione con el mundo exterior es necesario equiparlo de sensores que simulen los sentidos humanos.

En este contexto nace la necesidad de trabajar en la visión artificial. En la actualidad se trabaja en el desarrollo de visión artificial 3D en varias ramas:

- El láser 3D
- Cámaras del tipo Kinect, detallada en la Sección 3.1, la cual cuenta con tecnología CMOS y sensores de infrarrojos.
- Cámaras con tecnología CCD (dispositivo de cargas acopladas). Es un sistema matricial de píxeles en los que se recoge la información de la luz y se transforma en una señal analógica, cada señal analógica del dispositivo es digitalizada por un único conversor.

Las tecnologías son totalmente distintas por lo que en muchas ocasiones son compatibles y ninguna se impone por completo. En la Figura 1.2 se muestran dos robots con visión en 3D, uno con laser y otro con Kinect. Ambas tecnologías, láser 3D y cámaras de infrarrojos, trabajan con datos en forma de nubes de puntos, por lo que el software utilizado y la complejidad es parecida, lo que se puede destacar del láser 3D es su rango de trabajo, 30 metros, frente a los 3,5 de la Kinect. El punto fuerte de la Kinect es su precio, un 90% más barato que la tecnología láser. En estos tiempos, en los que la inversión en I+D es escasa, el desarrollo de soluciones económicas es la respuesta a corto plazo a las necesidades de la sociedad.

Figura 1.2: Robots Roomba con visión 3D

La visión con cámaras 3D es una tecnología cada vez mas extendida en la sociedad, la primera muestra es la cámara desarrollada por Microsoft para su consola Xbox, la Kinect, un hardware desarrollado para que el jugador sea el mando dentro del videojuego. Otras marcas como ASUS han desarrollado cámaras 3D con la misma idea, y tienden a acoplarla en sus portátiles (en lugar de la cámara webcam), otros

desarrolladores de hardware móvil como Samsung están trabajando en este tipo de cámaras 3D en sus dispositivos.

En el mundo de la robótica y la automatización, la Kinect se aplica para el desarrollo de humanoides y de robots tipo roomba, se trabaja en determinación de zonas transitables, reconocimiento facial, reconocimiento de personas, determinar tamaño de objetos con el fin de que sean usados por un humanoide, reconocimiento de objetos, etc. En la Figura1.3 se muestra el manipulador móvil de ocho grados de libertad MANFRED-2, desarrollado en el laboratorio de robótica de la UC3M [1], el robot es capaz de abrir puertas, evitar obstáculos y recoger y manipular objetos.

Figura 1.3: Manipulador móvil MANFRED-2

El reconocimiento de objetos es el tema que da objeto a este proyecto. Este documento está organizado en varios capítulos. El estado del arte se revisa en la Sección 2. Para llevar a cabo el proyecto se utilizará una cámara de las características Kinect, descrita en la Sección3.1, un ordenador con sistema operativo Ubuntu 11.0 y el software necesario para controlar la cámara, especificado en la Sección3.2. El reconocimiento de formas geométricas se realizará a partir de la captura de imágenes de la cámara Kinect, las imágenes serán tratadas como nubes de puntos, definido en la Sección4.2, y se abordaran con el algoritmo RANSAC, concretado en la Sección4.3, para encontrar modelos geométricos en ellas. Los modelos geométricos que se dan lugar en este trabajo son los prismas rectangulares, esferas y cilindros. Tras las pruebas experimentales se propondrá una aplicación en el contexto de nuevas tecnologías para

la terapia, descritos los experimentos y la propuesta de aplicación en la Sección5. Finalmente las principales conclusiones se detallan en la Sección6.

1.1. Objetivo

En esta sección se van a explicar los objetivos que se desean obtener con este proyecto. El objetivo principal se basa en el desarrollo de un software capaz de distinguir objetos a través de la cámara Kinect. El proyecto se divide en varias partes:

- Captura de imágenes desde el PC y la posterior conversión en nubes de puntos tratables en el programa.
- Definición de algoritmos capaces de identificar las formas, siendo las formas a tratar cilindros, esferas y prismas.
- Tratar la información de salida del software y una aplicación real.

Se ha planteado el objetivo de no solo diferenciar entre objetos conocidos si no de distinguir formas geométricas que caracterizan esos objetos. La opción de mostrar un objeto a la cámara y que esta lo reconociera posteriormente no es estable puesto que se busca un software robusto, que pudiera ser utilizado en diversas aplicaciones.

Para el desarrollo del programa necesario se ha recurrido al lenguaje de programación C++ por dos motivos, la mayoría de desarrolladores de software que están trabajando con Kinect utilizan las librerías de software libre PCL desarrolladas en lenguaje C++ y porque el lenguaje C++ es un lenguaje muy extendido que permite que mi programa sea insertado en programas ya desarrollados para otras aplicaciones, por ejemplo controladores para el movimiento de un robot.

El propósito de la separación en tres partes del desarrollo del programa es con vistas a posibles ampliaciones y para dar versatilidad al programa, permitiendo que las imágenes analizadas puedan ser capturadas en el momento o se utilicen imágenes de archivo. La parte de analizado de la imagen podrá ser ampliada para introducir mas formas geométricas. La salida del programa es un valor de un entero (int) que indica el tipo de forma y es la entrada a cualquier aplicación.

Para conseguir realizar todo lo mencionado anteriormente, en la concepción del programa se plantearon las siguientes especificaciones:

- La Kinect se sitúa sobre su propia caja (38x12x15 cm.) con una inclinación del cuerpo de la cámara de 5,4º.
- El espacio a analizar será el volumen de un prisma de (60x60x43cm).
- Los objetos se analizarán en unidades sobre superficies planas (mesa, suelo, etc.).

2. Estado del arte

Los puntos clave de este trabajo son las librerías PCL, la cámara en tres dimensiones Kinect y el algoritmo RANSAC. Por lo que enfocaré el estado del arte en estos tres puntos.

Una tarea importante para un robot es identificar su entorno y para ello la visión es uno de los sentidos clave, las nuevas tecnologías otorgan habilidades de visión al robot que superan las del ser humano. Por esta razón la investigación entorno a la visión artificial, y sobre todo a la visión en 3D, es un referente en la investigación dentro del mundo de la robótica. Un ejemplo claro es la utilización del dispositivo de Microsoft, Kinect. Este dispositivo dotado de una cámara, un sensor infrarrojos que otorgan sentido de profundidad y una serie de arrays de micrófonos ya tiene aplicación en el mundo real y no solo en el mundo del entretenimiento.

Figura 2.1: Cirujano utilizando la aplicación TedCas

Una de las aplicaciones hecha realidad es la del control sin contacto de las pantallas de un quirófano. Esta aplicación esta desarrollada por una empresa española, TedCas [2] (2011) seleccionada por Microsoft como una de las mejores aplicaciones para Kinect a nivel mundial, su fin es disminuir el número de infecciones de los pacientes durante las operaciones. En un quirófano normal los médicos tienen acceso a ordenadores con pantallas en las que pueden leer información sobre el paciente, imágenes radiológicas, modelos 3D, etc. Debido al contacto con estas nuevas tecnologías dentro de los quirófanos cada año 6 millones de pacientes adquieren una infección en el hospital y hasta 200 mil de ellos mueren. Estas infecciones, además, cuestan billones cada año a los gobiernos. En la Figura2.1 un cirujano utilizando la aplicación. Gracias a la tecnología de la cámara Kinect, los cirujanos pueden trabajar con los ordenadores sin contacto, solo con comandos por voz y con el movimiento de sus manos.

Las librerías PCL, la herramienta de trabajo en este proyecto, es una herramienta usada por desarrolladores de robots para su navegación por complejos entornos en 3D.

Esta tecnología es utilizada desde proyectos de código abierto de universidades y aficionados, como el proyecto Bilibot [3], hasta empresas y centros de investigación como WillowGarage [4].

Bilibot es un proyecto del Instituto Tecnológico de Massachusetts, es un robot que venden a universidades e instituciones a bajo coste para su uso en investigación.

Figura 2.2: Robot Bilibot

Bilibot viene equipado con un ordenador Core i3 4 GB de RAM, un sensor Kinect y un robot de iRobot. El software que utiliza es de código abierto: sistema operativo Ubuntu; ROS, son librerías y herramientas para el desarrollo de aplicaciones para robots ; OpenNI, que es la comunicación entre el sensor Kinect y el resto del software; PCL, las librerías que permiten al conjunto Bilibot procesar a tiempo real el gran volumen de datos que proporciona la cámara Kinect.

Willow Garage es una incubadora de empresas y un laboratorio de investigación importante en el mundo de la robótica, se dedica al desarrollo de robots como PR2 y Turtlebot (Con la misma filosofía que Bilibot) trabajando con software de código abierto como ROS o PCL.

Se puede destacar PR2, un robot de tamaño humano, con ruedas y dos brazos, cada brazo tiene tres articulaciones que pueden ser llamadas hombro, codo y pinza con cuatro, tres y un grados de libertad respectivamente. Tiene comunicación Ethernet, wifi y Bluetooth. Usa como sensores varias cámaras en su cabeza donde destaca la Kinect, en cada brazo dispone de otra cámara, y un láser en su base y otro debajo de su cabeza.

Figura 2.3: Robot PR2

Este robot, Figura2.3, está preparado para estar en contacto con humanos, en tareas repetitivas, para tareas del hogar y para la ayuda a personas dependientes. Gracias al software libre que incluye: ROS, PCL, etc. Permite en la actualidad que investigadores de todo el mundo busquen aplicaciones reales a este hardware tan potente.

El algoritmo RANSAC es actualmente objeto de investigación y aplicación en diversas materias. Se puede destacar el trabajo de la universidad de Zurich, destacan investigaciones como la del profesor Scaramuzza [5] y en el campo del reconocimiento de objetos, la investigación de ChavdarPapazov [6].

El profesor Scaramuzza presenta un nuevo método para estimar el movimiento de un vehículo, el mostrado en la Figura2.4, con la información de una sola cámara. En el mundo de la visión y del tratamiento de puntos, el principal problema es detectar los puntos que son ruido de los puntos que son importantes. En los últimos años un método utilizado ha sido el de 5 puntos de RANSAC, un algoritmo que necesita un mínimo de 5 puntos para estimar la hipótesis del modelo como correcta, esto obliga ha realizar cientos de iteraciones para encontrar un conjunto de inliers. Se plantea el uso de una sola correspondencia para la estimación del movimiento con el uso de dos algoritmos para eliminar los valores extremos, 1 punto de RANSAC y los votantes de histograma. Para ello Scaramuzza muestra una aplicación del método de odometría visual mediante la recuperación de una trayectoria de 3 Km. en un entorno urbano desordenado y en tiempo real.

Figura 2.4: Estimación de movimiento de un vehículo con algoritmo RANSAC

Papazov presenta un algoritmo eficiente para el reconocimiento de objetos en 3D, en presencia de ruido y de datos con falta de información, como pueden ser sombras o puntos muertos. El método utiliza un descriptor robusto de geometría y el algoritmo RANSAC. Papazov supone que cada objeto esta representado por un modelo que consta de un conjunto de puntos con sus correspondientes normales, al igual que el usado en el presente trabajo. El método reconoce características del modelo y calcula su posición y orientación en la escena, este método es probado en una variedad de conjuntos en los que las nubes de puntos, Figura2.5, del objeto solo muestran parte del mismo.

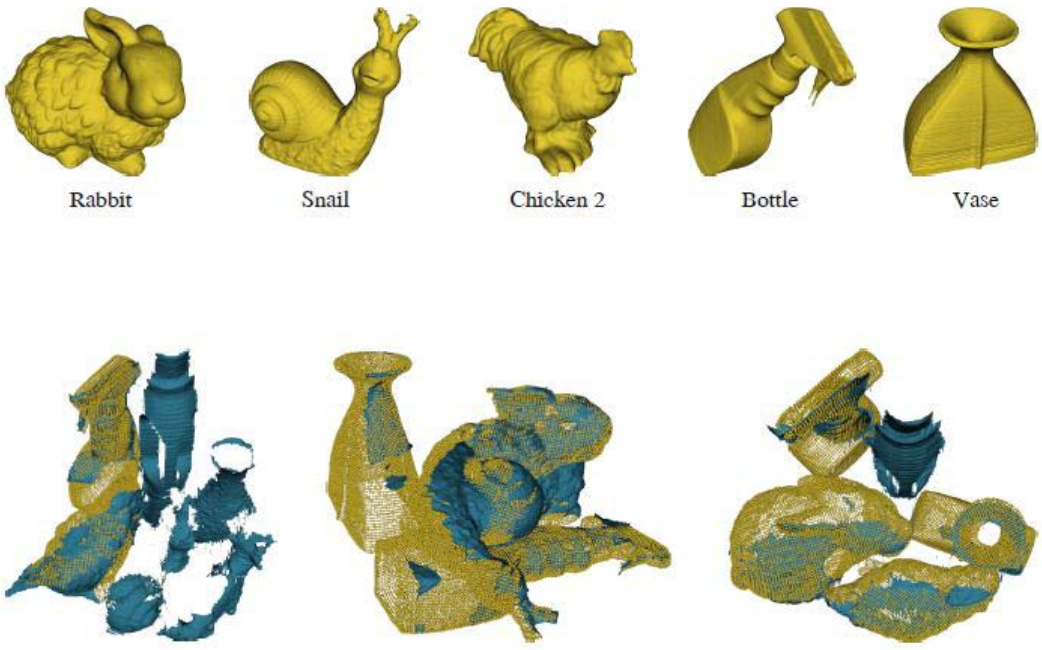


Figura 2.5:Nubes de puntos de Papazov para el algoritmo RANSAC

3. Herramientas para el desarrollo del trabajo

En el presente capitulo se explica los principales instrumentos necesarios para la elaboración del trabajo, la cámara Kinect para la captura de imágenes y las librerías PCL.

3.1. Kinect

La cámara Kinect se puede dividir en dos partes principales una base y un cuerpo alargado unidos por un motor. En el cuerpo de la Kinect encontramos una cámara RGB (Red Green Blue), un conjunto de micrófonos y un sensor de profundidad, en la Figura3.1 se muestran los distintos elementos. Gracias al sensor de profundidad la información recibida es en tres dimensiones. Las dimensiones de la cámara son 28,3 cm. de largo 7,5 cm. de ancho y 7 cm. de alto.

Figura 3.1: *Características Kinect*

La cámara RGB, detecta los tres colores primarios y tras un código (Rojo, Verde, Azul), que indica el porcentaje de cada color, se determina así el color del píxel, la cámara es capaz de capturar 30 imágenes por segundo. La cámara tiene 640x480 píxel en 32-bit.

Figura 3.2: Interior Kinect 1

El sensor de profundidad se compone por un puerto infrarrojos, y un sensor CMOS (Semiconductor de Óxido de Metal Complementario) monocromo. El sensor tiene un rango de entre 1,2 y 3,5 metros de profundidad y un rango angular de 57 grados en horizontal y 47 grados en vertical. El motor acoplado a la base permite un rango de movimiento entre 27 grados positivos y negativos. La tecnología CMOS se basa en una red de semiconductores distribuidos en forma de matriz, en cada celda de la matriz, en función de la cantidad de luz, se acumulará una carga eléctrica. La cámara deproximidad tiene 16 bit y una matriz de 320x240 píxel. El emisor de infrarrojos emite luz la cual tras rebotar en los objetos será captada por el sensor con tecnología CMOS. El conjunto del sensor de proximidad proporciona un cuarto valor al dato de cada píxel, caracterizando de este modo un dato de la Kinect en RGBX.

Figura 3.3: Interior Kinect 2

La alimentación de la cámara se basa en una conexión USB y alimentación de 12v y 1,08 A en corriente continua.

3.2. Librerías PCL

PCL es un proyecto que inició Willow Garage en marzo de 2011 para el procesamiento de nubes de puntos en 3D. El proyecto ha continuado en todo el mundo con la colaboración de científicos, empresas y universidades gracias a que el software es un código abierto y gratuito. PCL es multiplataforma, y ha sido compilado con éxito y desplegado en Linux, MacOS, Windows y Android / iOS.

Las librerías que componen PCL están preparadas para trabajar con nubes de puntos. Una nube de puntos es una estructura de datos en 4D en la que encontramos cuatro datos, las coordenadas X Y Z de cada punto y el color en formato RGB, esta nube de puntos se puede adquirir a partir de sensores como cámaras 3D o láser, en el mercado encontramos hardware como PrimeSensor 3D, XTionPRO Asus y el Kinect de Microsoft que ocupa lugar en este proyecto.

Los archivos que contendrán las nubes de puntos son del formato .PCD, un formato original de PCL basado en formatos como .PLY, .STL, .OBJ o .X3D. Cada archivo .PCD, como el mostrado en la Figura3.4, incluye una cabecera con información:

- Versión.
- Nombre de cada dimensión (x, y, z, rgb, normal_x, normal_y, etc.).
- Tamaño (4bytes).
- Tipo (F, float).
- Número de elementos que ocupa cada dimensión (normalmente 1).
- Ancho.
- Alto (si es 1 los puntos están desordenados y el ancho es el total de la nube).
- La orientación en la que se han adquirido los datos.
- Número total de puntos.
- Tipo de código usado en el archivo (ASCII o binario).

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
```

```
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
0.81915 0.32 0 4.2108e+06
0.97192 0.278 0 4.2108e+06
0.944 0.29474 0 4.2108e+06
0.98111 0.24247 0 4.2108e+06
0.93655 0.26143 0 4.2108e+06
0.91631 0.27442 0 4.2108e+06
0.81921 0.29315 0 4.2108e+06
0.90701 0.24109 0 4.2108e+06
.
.
.
```

Figura 3.4: Archivo .pcd

En el presente proyecto se utilizarán clases de distintas librerías de PCL:

- **I/O**, librería con clases y funciones para la lectura y escritura de los archivos .PCD, y la captura de datos por la cámara Kinect.
- **Filtres**, que ofrece clases para el filtrado (como "PassThrough" Sección4.2 adaptación del entorno), de puntos que no siguen una norma o especificación.
- **Features,** es una librería que contiene funciones para estimar zonas que cumplen especificaciones, como la vecindad de puntos, y pueden ofrecer información sobre las normales en superficies.
- **Sample consensus**, es una librería útil para el método de tratamiento de los puntos que se ha elegido en este trabajo, RANSAC, porque incluye modelos como cilindros, planos, conos, etc.
- **Segmentation**, contiene algoritmos de segmentación como Random Sample Consensus,Least Median of Squares, Progressive Sample Consensus, etc.
- **Visualization** es la librería que contiene "viewer" que permite la visualización de los archivos .PCD por parte de los usuarios, ordenando los puntos y permitiendo visualizar los colores, la distancia de los puntos en cada eje respecto al origen.

Para el intercambio de información entre el sistema operativo y la cámara es necesario instalar OpenNi, es un proyecto de código abierto enfocado a la integración de los sensores de audio, video y sensor de profundidad de Kinect con las librerías PCL.

El lenguaje de programación usado en las librerías PCL es C++ [7,8], un lenguaje orientado a objetos. La programación orientada a objetos (POO) fue desarrollada, esencialmente, por las limitaciones que tiene la programación estructurada C. La programación en C se basa en funciones, las cuales tienen un propósito claramente

definido, la información se pasa entre funciones utilizando parámetros y las funciones pueden tener datos locales a los cuales no se pueden acceder fuera del ámbito del procedimiento.

En laprogramación estructurada las funciones se llaman unas a otras y tienen como objetivo el algoritmo y no el uso de los datos. La POO, tiene como objetivo modelizar la realidad, las clases describen una realidad, las características (atributos de la clase) y las funciones (métodos de la clase), y cada objeto es una realidad en concreta. La clase atleta puede tener el atributo altura, y el método correr, y un objeto de esa clase puede ser Usain Bolt, un atleta en concreto, que puede correr y tiene una altura en particular.

Las clases más relevantes utilizadas en este trabajo son las siguientes:

PassThrough

Las funciones públicas de la clase "PassThrough"[9]son utilizadas en la Sección 4.2:

- "voidsetFilterFieldName(conststd::string &field_name)"
- "voidsetFilterLimits (const float float&limit_min, const float &limit_max)"

La primera sirve para proporcionar el nombre del ejeen el que se filtran los datos. La segunda se usa siempre en unión con la primera y en ella se determina los límites. Fuera de este intervalo los puntos serán descartados. El nombre del eje será una cadena de texto. Los límites de la función "setFilterLimits" deben de ser dos números decimales de tamaño 32 bits.

SACSegmentationFromNormals

La clase "SACSegmentationFromNormals"[10]está dedicada a los métodosSampleConsensus, incluido RANSAC, definido en la Sección 4.3, y para los modelos que requieran las normales de los datos de entrada. Las funciones utilizadas en el presente trabajo son:

- "voidsetInputNormals (constPointCloudNConstPtr&normals)"

Se utiliza para proporcionar el archivo que contiene la información de las normales del conjunto de datos XYZ.

- "voidsetNormalDistanceWeight (doubledistance_weight) "

Se establece el porcentaje entre 0 y 1 para dar el ángulo entre 0 y 90º entre las normales del punto y el plano que lo contiene.

- "void setModelType(intmodel)"

A través de un número entero se introduce el tipo de modelo para usar con el algoritmo RANSAC. En el trabajo se usan los siguientes modelos:

PERPENDICULAR_PLANE(9)
SPHERE(4)
CYLINDER (5)

- “setMethodType(int method)”

A través de un número entero se introduce el tipo de método, siendo RANSAC (0) el elegido para este trabajo.

- “setDistanceThreshold(doublethreshold)”

Con el valor de un número decimal se determina el porcentaje de concordancia entre el modelo y los datos de entrada.

- “setMaxIterations(intmax_iterations)”

Con un número entero se define el número máximo de iteraciones que realiza el algoritmo para dar un resultado.

- “setRadiusLimits(const double &min_radius, const double &max_radius)”

Sirve para fijar los límites mínimos y máximos permitidos a los modelos cilindro y esfera.

- “setInputCloud(constPointCloudConstPtr&cloud)”

Mediante esta función se introduce los datos de entrada mediante una nube de puntos.

- “segment(PointIndices&inliers, ModelCoefficients&model_coefficients)”

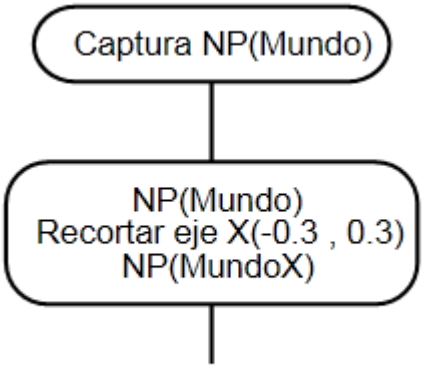
Devuelve el resultado del algoritmo en forma de índices que marcan el resultado y extrayéndolos de los datos de entrada nos dan la nube de puntos resultado.

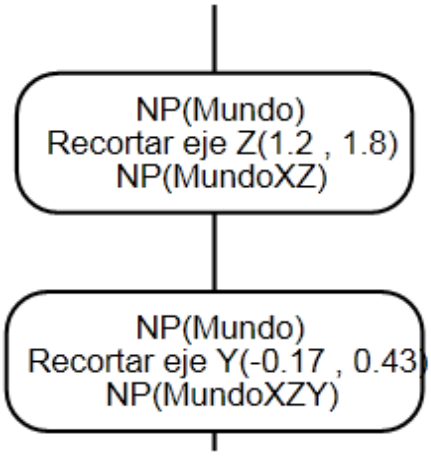
4. Reconocimiento de formas

En el presente capítulo se detalla el grueso del trabajo, en la Sección 4.1 se muestra un diagrama de flujo del programa desarrollado explicando desde la captura de nubes de puntos con la cámara hasta el resultado final del algoritmo, el cual nos indica la Figura geométrica que caracteriza el objeto. En la Sección 4.2 se detalla la adaptación del entorno, desde una imagen real, hasta una nube de puntos del tamaño especificado en el objetivo de la Sección 1 del presente trabajo, que contiene un objeto sobre una superficie plana. En presente capitulo se detallaran los algoritmos RANSAC y k-d Tree. Desde la Sección 4.5 a la Sección 4.8 se concretan los modelos geométricos plano, esfera, cilindro y el método para el reconocimiento de prismas. En estos subcapítulos se muestran imágenes reales capturadas por la cámara, y la manera de ajustar el algoritmo RANSAC a nuestras necesidades.

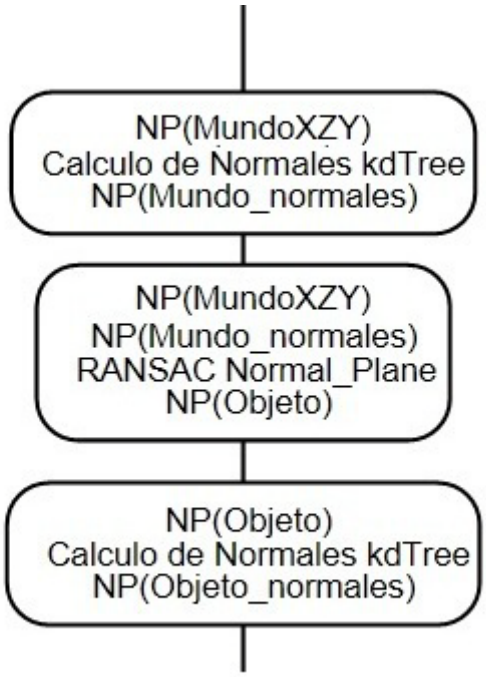
4.1. Diagrama de flujo

A continuación se presenta el diagrama de flujo del proceso completo de identificación de formas. En el trabajamos con nubes de puntos (NP), con el algoritmo k-d Tree, definido en la Sección 4.4, y con el algoritmo RANSAC y sus distintos modelos, aclarado en la Sección 4.3.
El primer paso es la captura de imágenes con los sensores de proximidad y la cámara RGB de la Kinect, la captura de la imagen se guardara en una nube de puntos (NP) llamada Mundo, en la que se encontrará el objeto a tratar con su entorno. Lo habitual será que la NP Mundo este compuesta por todos los objetos de una habitación y entre los 1,2m y 1,8 m de distancia (en el eje Z) de la Kinect se encuentre el objeto a evaluar. El objeto ocupará como máximo 60cm de alto y 60cm de ancho. Según estas especificaciones se procederá al filtrado de la NP como se detalla en la Sección 4.2 hasta obtener una nube de puntos compuesta por el objeto y una superficie plana la cual se llamará MundoXZY.

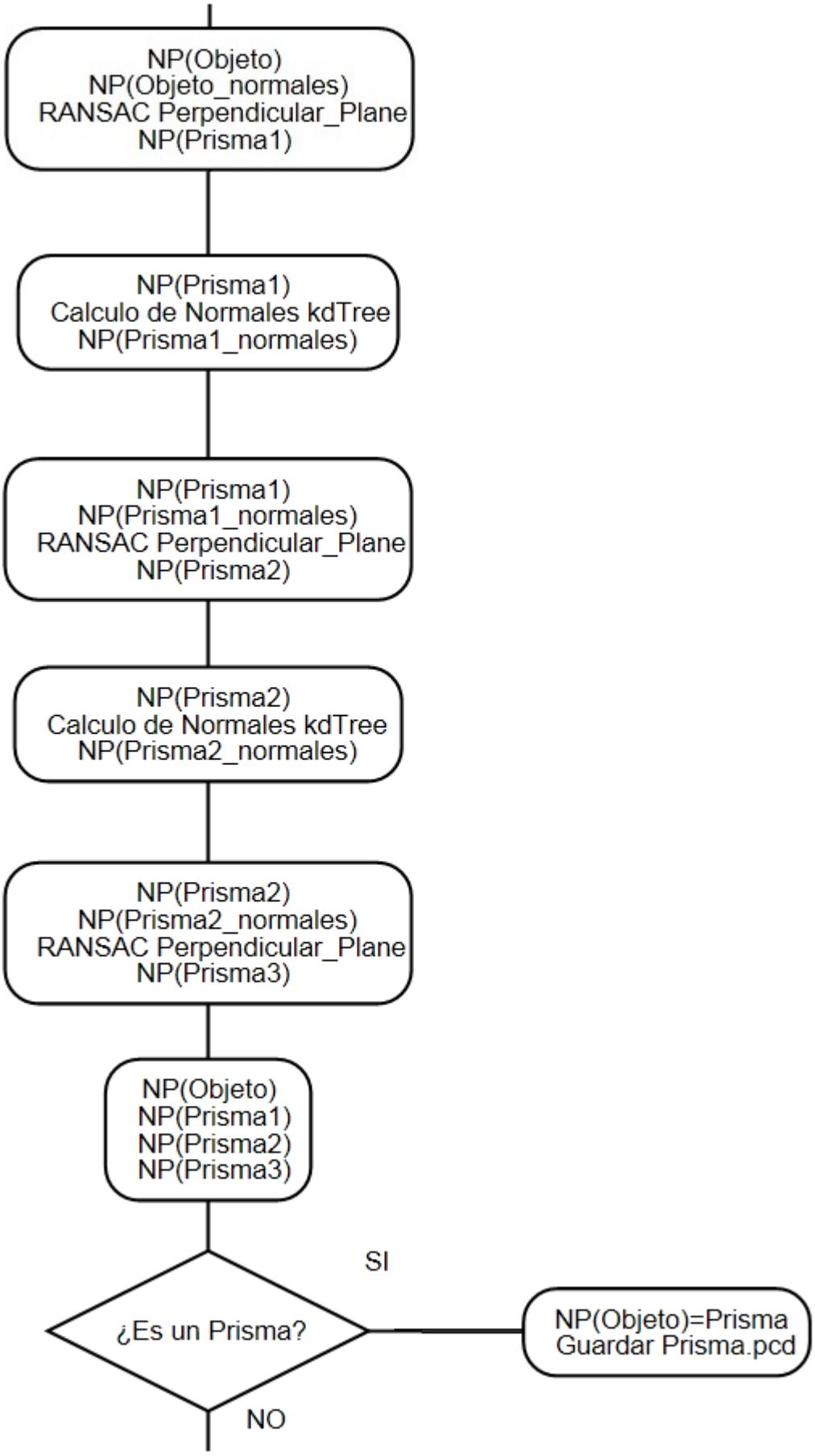




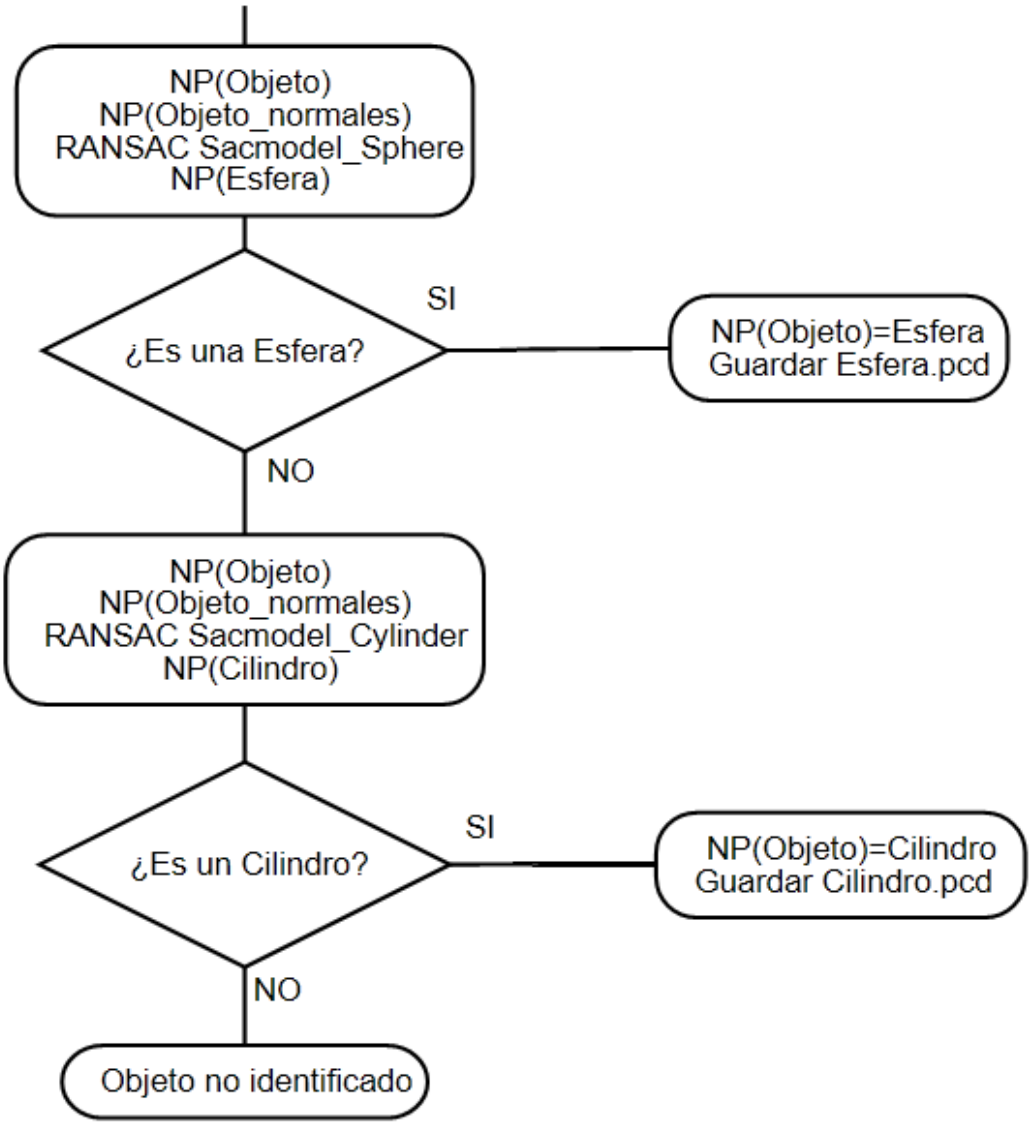
El paso siguiente a preparar la imagen es extraer el objeto de la nube de puntos recortada. Para ello es necesario calcular las normales de la nube de puntos, para mejorar el calculo de las normales se utilizará el algoritmo k-d Tree para acelerar la búsqueda de vecinos de cada punto y con ellos calcular las normales de cada punto, proceso explicado en la Sección 4.4. A continuación se aplica el algoritmo RANSAC teniendo como entradas la nube de puntos MundoXZY, las normales de la misma y el modelo de un plano. En este paso el resultado será la nube de puntos correspondiente al objeto, este paso está explicado en la Sección 4.5.



La primera suposición es que el objeto es un prisma rectangular, para comprobarlo es necesario extraer de la imagen los tres planos visibles que pueden formar dicho prisma, y tras analizarlos junto a la imagen del objeto se puede determinar si el objeto corresponde o no a un prisma. A través del cálculo de normales y del algoritmo RANSAC junto a un modelo de un plano perpendicular se determinaran dichos planos. El procedimiento en detalle se explica en la Sección 4.6.


NP(Objeto)
NP(Objeto_normales)
RANSAC Perpendicular_Plane
NP(Prisma1)
NP(Prisma1)
Calculo de Normales kdTree
NP(Prisma1_normales)
NP(Prisma1)
NP(Prisma1_normales)
RANSAC Perpendicular_Plane
NP(Prisma2)
NP(Prisma2)
Calculo de Normales kdTree
NP(Prisma2_normales)
NP(Prisma2)
NP(Prisma2_normales)
RANSAC Perpendicular_Plane
NP(Prisma3)
NP(Objeto)
NP(Prisma1)
NP(Prisma2)
NP(Prisma3)
SI
¿Es un Prisma?
NP(Objeto)=Prisma
Guardar Prisma.pcd
NO

Si el objeto no ha sido determinado como un prisma rectangular, el siguiente paso será comprobar si el objeto extraído, de la nube de puntos Mundo, es una esfera para ello se seguirán las indicaciones de la Sección 4.7. En este paso se aplicará el algoritmo RANSAC con el modelo Sphere, y las nubes de puntos de entrada serán el objeto y las normales del mismo. Si el objeto no corresponde al modelo esfera, se analizará con el modelo cilíndrico, detallado en la Sección 4.8 del presente capítulo. En el caso en el cual el objeto no corresponde a un prisma rectangular, ni a una esfera ni a un cilindro el programa devolverá un mensaje en el que indica que la forma del objeto no ha sido identificada.



4.2. Adaptación del entorno

Para cumplir los objetivos planteados en el proyecto, teniendo en cuenta las especificaciones, es necesario describir el entorno en el que trabaja la cámara. Se parte de la base en la que el objetivo de la cámara se situará a 43 cmde altura de la superficie plana en la que se encuentra el objeto a analizar. Se quiere analizar un objeto que sea, como máximo, igual de alto que el objetivo de la cámara, o lo que es lo mismo, la Kinect situada encima de su propia caja.

La cámara tiene un rango de trabajo entre 1,2 y 3,5 metros, por lo que si se quiere trabajar en una superficie de 60cm las cotas de trabajo estarán entre los 1,2 y 1,8 metros. En la Figura4.1 se representa la zona de trabajo con el objeto (estrella) en su interior, y la cámara (circunferencia) entre 1,2 y 1,8 metros del objeto.

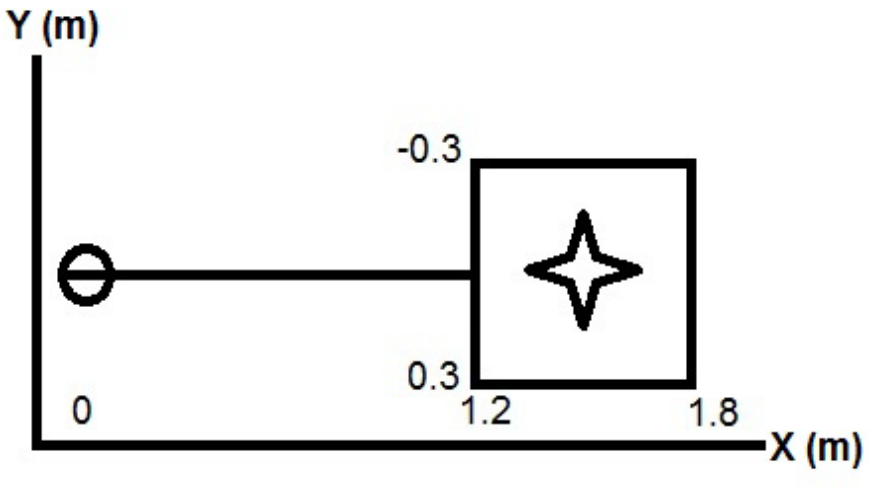


Figura 4.1:Esquema zona de trabajo Y-X

Según las especificaciones técnicas la cámara tiene un rango de 54º en vertical, y el motor posiciona el cuerpo en 10 posiciones, por lo que cada posición implica una inclinación de 5,4º. Para mejorar la captura de imágenes se inclinara el cuerpo de la cámara 5,4º. Hay que tener en cuenta que, debido a esta inclinación, el volumen de prisma en el que se quiere trabajar ya no tendrá las dimensiones de (60x60x43cm) entre las coordenadas 1,2 y 1,8 del eje X en las que se quiere trabajar.

Sabiendo la inclinación de la cámara y la distancia del punto en el eje X con trigonometría se calcula la altura Z.

$$\text{Tan}\,(5{,}4º) = Z / X$$

Por lo que a 1,2 metros de la cámara tendremos una altura de 31,66 cm. y a 1,8 metros tan solo 26cm, representación en la Figura4.2, 17cm menos de los que se plantean en las especificaciones.

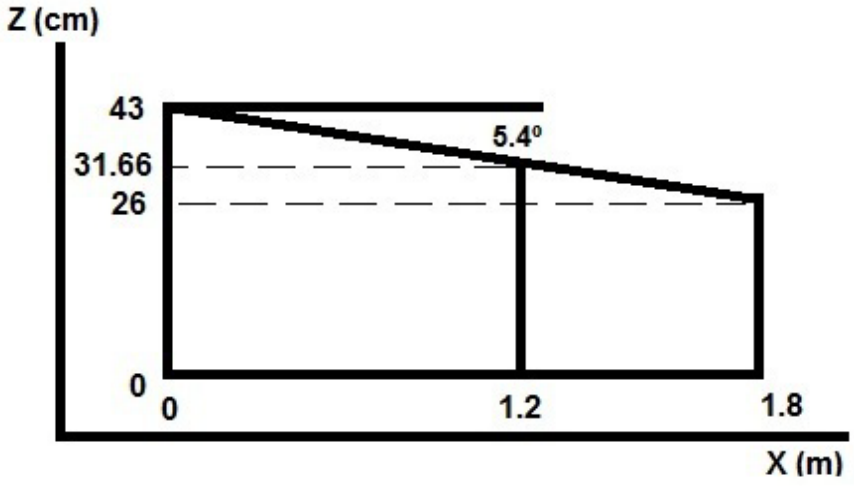


Figura 4.2:Esquema zona de trabajo Z-X 1

Se utilizará el software para poder resolver este problema. La imagen original de la Kinect será recortada en la forma del prisma deseado, con unas dimensiones de 60x60 cm. de base, y una altura de 60cm, que debido a la inclinación corresponderá a 43cm en el punto mas alejado y 48,46cm en el primer punto de la imagen como se observa en la siguiente figura.

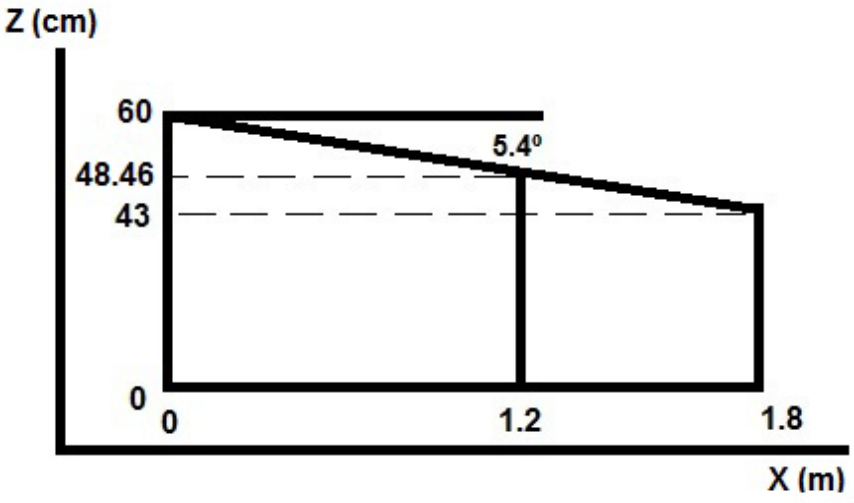


Figura 4.3:Esquema zona de trabajo Z-X 2

Para recortar la nube de puntos que nos captura la cámara utilizaremos la clase "PassThrough" la cual nos permite recortar la imagen en las coordenadas (X Y Z). Para el controlador de la Kinect el punto 0,0 se encuentra en el punto medio del cuerpo. Como se indica en la Figura4.4 los valores positivos en el eje Z serán desde la Kinect hacia delante. En el caso del eje X los valores positivos serán a la izquierda y los

negativos a la derecha respecto de la visión de la cámara. Para el eje Y los valores positivos serán en la parte inferior y los negativos en la superior.

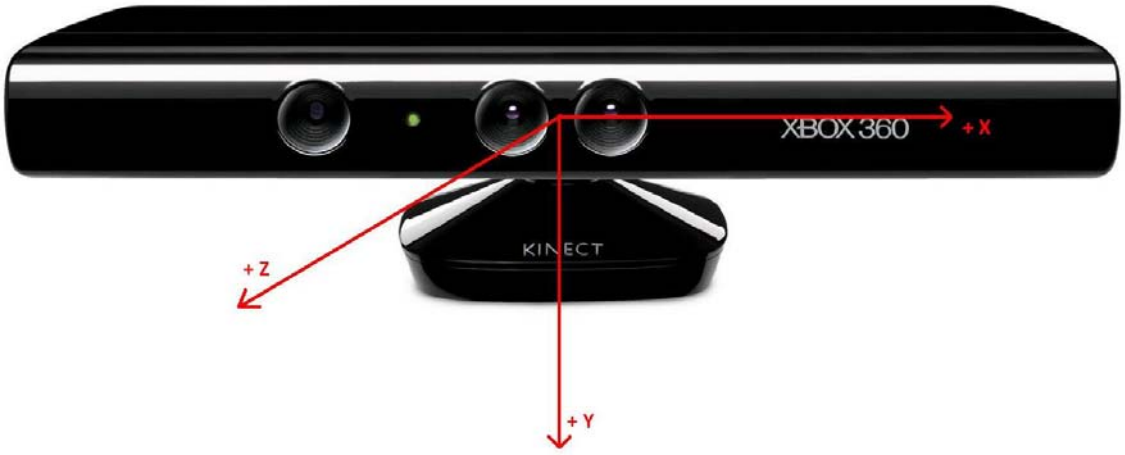


Figura 4.4: Coordenadas en Kinect

Se deben marcar los límites en cada coordenada quedando en la nueva nube de puntos el interior de dichos límites. Según este criterio de signos, el corte será del siguiente modo:

- X [-0,3 0,3]
- Y [-0,17 0,43]
- Z [1,2 1,8]

Para trabajar con la clase “PassThrough” se necesita crear tres objetos, uno para cada eje que se quiere recortar. Como ejemplo el caso del eje X:

Se crea el objeto “pasx” y se utiliza para trabajar con cada función:

- “pcl::PassThrough<pcl::PointXYZRGBA> pasx;”

Se introduce la nube original:

- “pasx.setInputCloud (cloud);”

Se indica que trabajamos en el eje X:

- “pasx.setFilterFieldName ("x");”

Se imponen los valores de los límites:

- “pasx.setFilterLimits (-0,3, 0,3);”

Se obtiene la nube recortada en el eje X, que será la nube de entrada para el siguiente corte:

- “pasx.filter (*cloud_filteredx);”

A continuación se muestran imágenes obtenidas por la cámara:

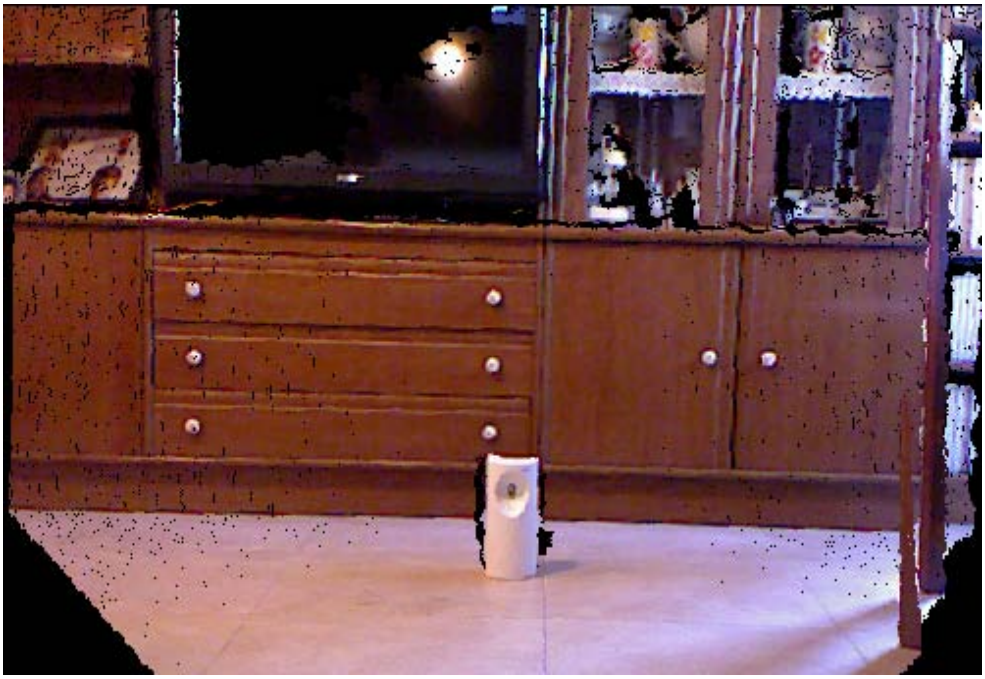


Figura 4.5:Captura original nube de puntos en color

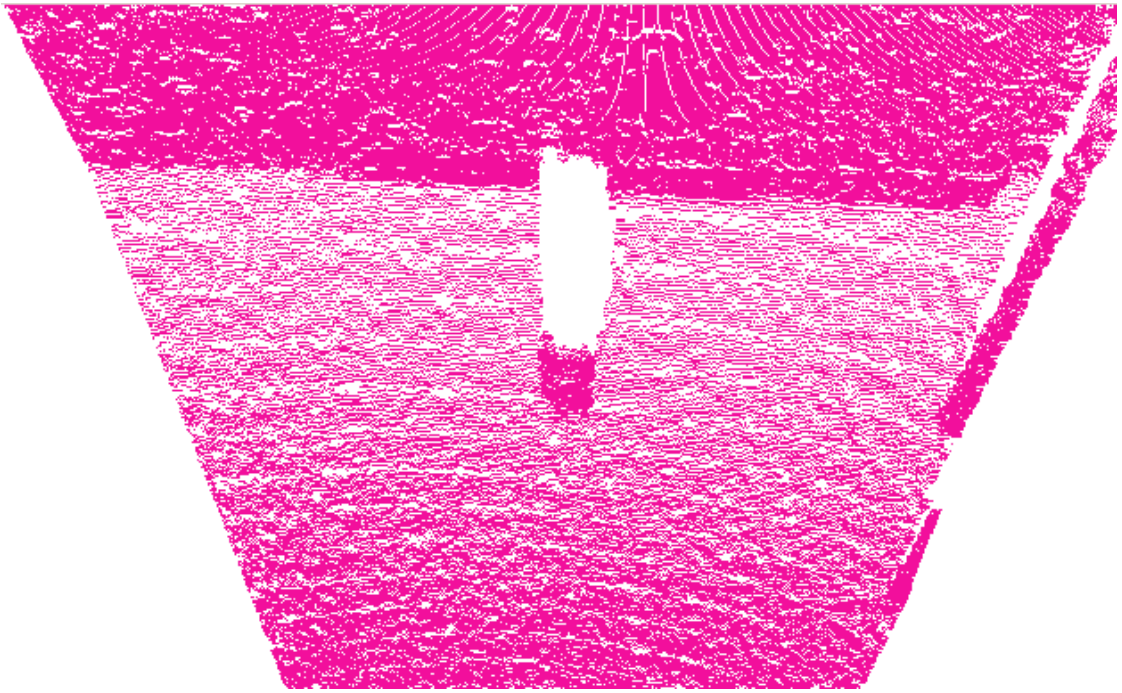


Figura 4.6:Nube de puntos vista superior

En la Figura4.5 la imagen es la original obtenida por la cámara, es una representación de nube de puntos en color. La Figura4.6 representa la misma escena pero desde un ángulo superior y sin representar los puntos a color, destaca la sombra blanca (sin puntos) que deja el objeto.

Después de recortar la nube de puntos en las tres coordenadas, se obtiene la nube que será la entrada para los procesos siguientes, Figura4.7. En la imagen debería quedar una superficie plana de 0,36m2 con el objeto a tratar.

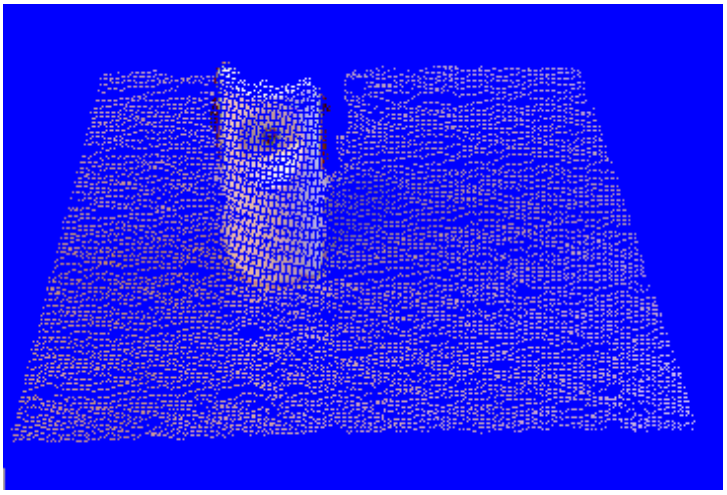


Figura 4.7:Nube de puntos recortada

4.3. Algoritmo RANSAC

El método elegido para el tratamiento de datos es RANSAC [11] (Random Simple Consensus) o lo que es lo mismo, consenso de muestra aleatoria. Este método a través de unos datos de entrada y un modelo matemático devuelve un conjunto de datos como resultado, inliers. Este algoritmo fue publicado por Fischler y Bolles en 1981 [12].

Todos los datos se componen de inliers y outliers, siendo los inliers un modelo matemático y unos parámetros y los outliers los datos que no encajan en el modelo.

Este algoritmo aumenta sus probabilidades de acierto en relación al número de iteraciones. El método incluye una serie de parámetros que aseguran que los inliers escogidos son los correctos y siguen un modelo correcto. Con otros métodos como el de mínimos cuadrados los datos que se incluyen en el estudio del método matemático son todos y no solo los inliers.

La entrada del algoritmo RANSAC son los datos originales, un modelo matemático parametrizado el cual sirve para ajustar en las iteraciones, también se incluye una serie de parámetros de confianza. En cada iteración el modelo devuelve como resultado unos inliers hipotéticos.

En el caso de este proyecto los datos de entrada son una nube de puntos, y los de salida otra nube que sigue un modelo matemático. De todos los datos de la nube de puntos, algunos siguen el modelo matemático y estos serán los inliers hipotéticos. El resto de puntos que no son inliers se vuelven a probar contra el modelo y si algún punto se ajusta al modelo matemático se incluirá en los inliers hipotéticos. Se considera que la estimación del modelo ha sido suficientemente buena si muchos puntos de la nube inicial han sido declarados inliers hipotéticos. El modelo vuelve a ser estimado con todos los inliers hipotéticos y no solo con los primeros seleccionados y finalmente se evalúa la estimación del error entre los inliers hipotéticos y el modelo. Estas iteraciones se repiten determinado número máximo de veces y se mantienen los inliers con menor error contra el modelo.

El algoritmo RANSAC es muy robusto y puede dar un buen resultado con unos datos de entrada repletos de extremos, o datos no inliers. Por otro lado presenta una desventaja por estar el número de iteraciones limitado, quizás en el numero de iteraciones especificado el resultado que se muestra no es el mas óptimo.

Una desventaja es que RANSAC sólo puede estimar un modelo para un conjunto de datos en particular. Por lo que es necesario realizar tantas pruebas como modelos se quieran analizar.

Para un modelo RANSAC general encontramos las siguientes variables de entrada:

- Datos de entrada, en nuestro caso una nube de puntos.
- Modelo, que puede ser cilindro, esfera, plano, circunferencia, etc.
- **n** el número mínimo de datos para ajustar el modelo.
- **k** el número de iteraciones realizadas por el algoritmo.
- **t** un valor umbral que indica cuando los datos se ajustan al modelo.
- **d** el número de valores de los datos necesario para afirmar que los datos de entrada se ajustan a un modelo.

Las variables de salida serán las siguiente:

- **Best_model** los parámetros del modelo que mejor se ajustan a los datos.
- **Best_consensus_set** los puntos de los datos de entrada que han sido estimados como mejores frente al modelo.
- **Best_error** el error de este modelo relativo a la entrada.

Se ha generado una nube de 5000 puntos aleatoria, se llamará Esfera_verde y está representada en la Figura4.8, con una esfera rodeada de puntos, para probar en ella distintos modelos geométricos de RANSAC, un plano, una línea, un círculo, un cilindro y una esfera.

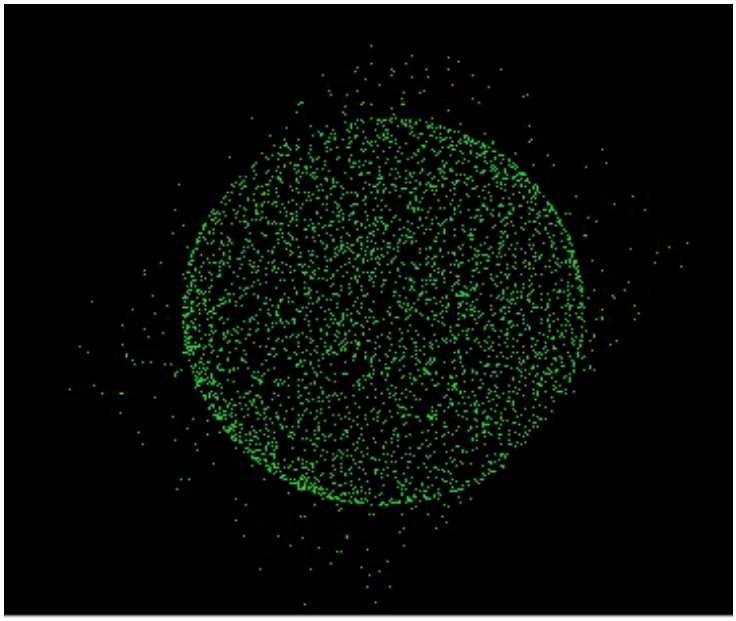


Figura 4.8:Esfera_verde de 5000 puntos

De los datos generales de entrada del algoritmo introduciremos:

- Datos de entrada: Esfera_verde.
- Modelo: Plano, línea, circulo 2D, cilindro y esfera.
- **k**: 1000.
- **t**: 3,36505e-317 es el valor mínimo, el cual indica que los datos de entrada se ajustan al 100% al modelo.

También se puede introducir el valor mínimo y máximo del radio en el que queremos trabajar.

Para este algoritmo también es necesario introducir como entrada las normales de la nube de puntos original.

Los resultados con el valor t mínimo son los siguientes:

- Plano: Solo tres puntos forman un plano, es decir no hay plano.
- Línea: No hay inliers.
- Circulo 2D: Solo tres puntos pertenecen a inliers.
- Cilindro: Solo tres puntos pertenecen a inliers.
- Esfera: Los inliers son 2664 puntos de 5000, representada en la Figura4.9.

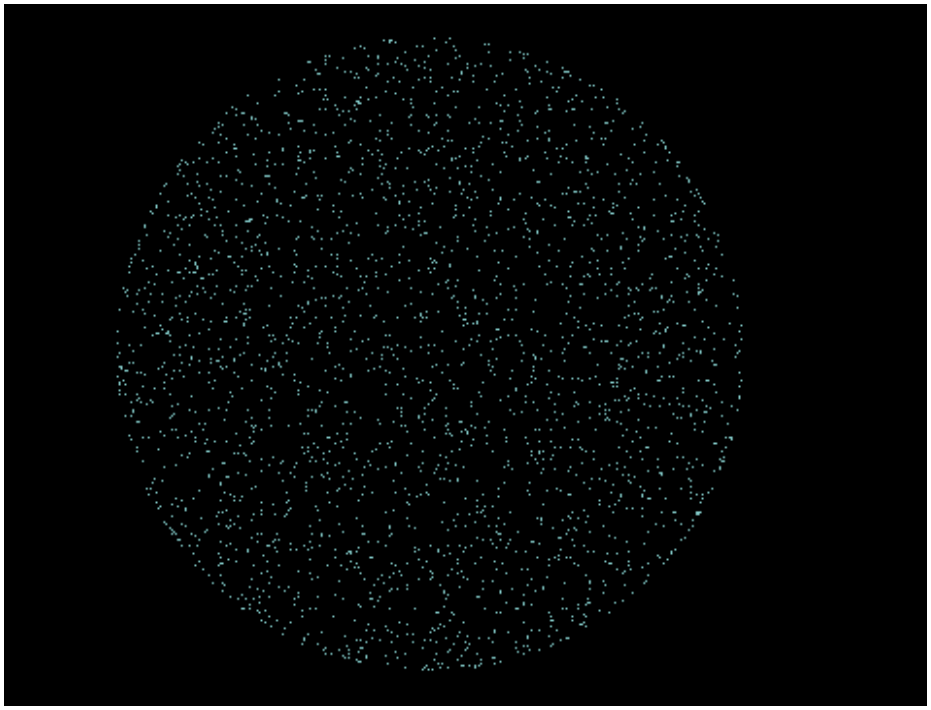


Figura 4.9:Esfera_verde con modelo esfera prueba 1

Si se aumenta el umbral (t) los resultados obtenidos no serán el modelo al 100% pero ayuda para ver el poder del algoritmo.

T=0,05.

- Plano: 327 puntos forman el resultado, un plano con apariencia de casquete esférico que se puede observar en la Figura4.10.

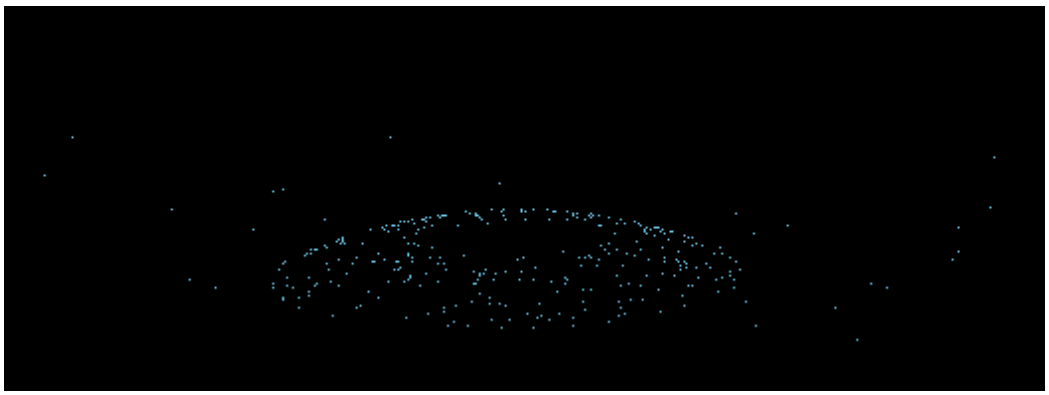


Figura4.10:Esfera_verde con modelo plano

- Línea: Una parte de una circunferencia de la esfera de 48 puntos forma una línea con este umbral.

- Circulo 2D: 771 puntos de 5000 forman un círculo en 2D, los resultados se muestran en las Figuras 4.11 y 4.12.

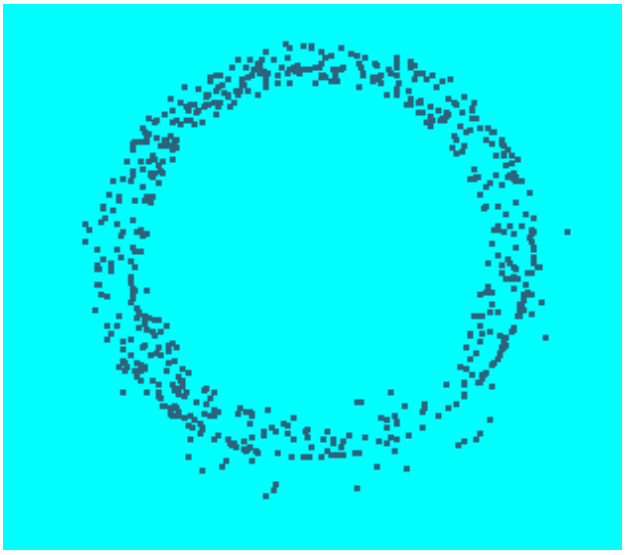


Figura4.11:Esfera_verde con modelo circulo 2D vista 1

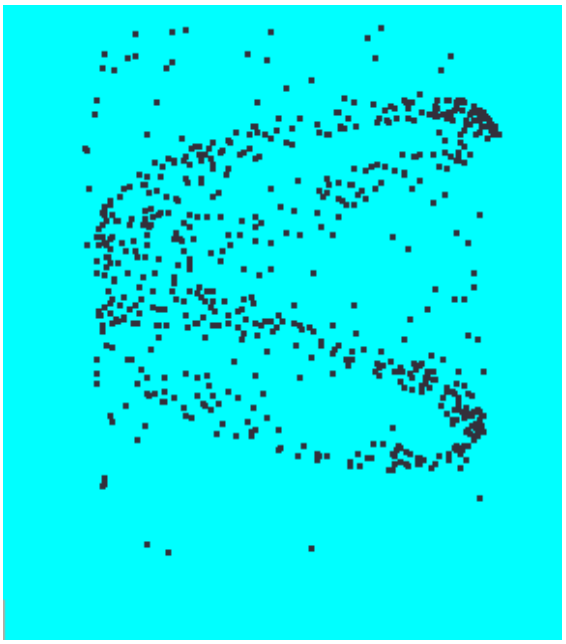


Figura4.12:Esfera_verde con modelo circulo 2D vista 2

- Cilindro: Los inliers según el algoritmo con un modelo de un cilindro son 172 puntos. Se puede visualizar en la Figura4.13.

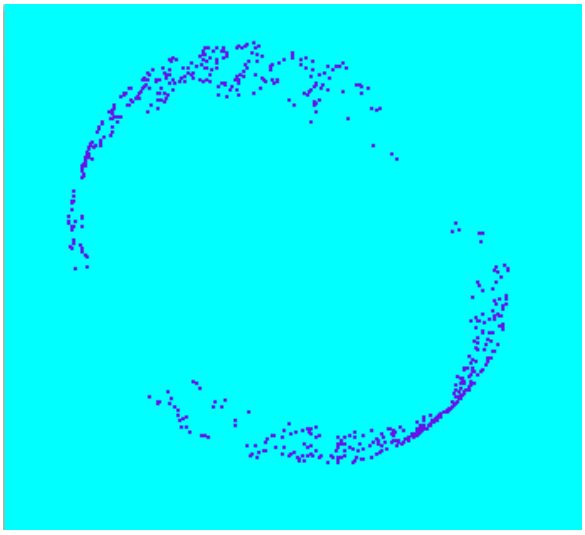


Figura4.13:Esfera_verde con modelo cilindro

- Esfera: Los inliers son 3284, 620 más que con el algoritmo en un umbral del 100%. El resultado se puede considerar muy bueno y como ventaja tenemos una nube de puntos más grande, una ventaja al representar objetos y al hacer filtrados, cuantos más puntos mas claro es para el ojo humano.

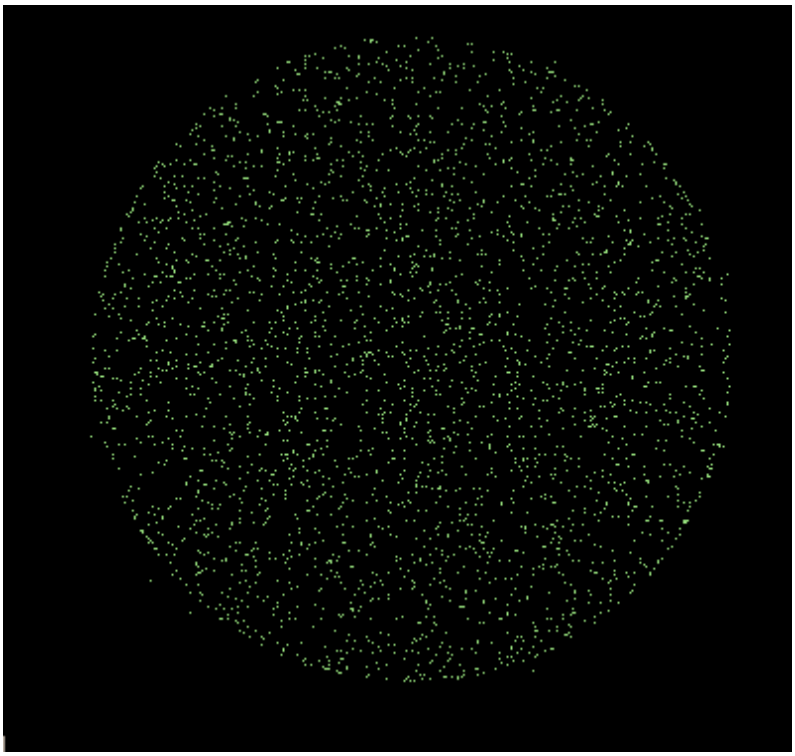


Figura4.14:Esfera_verde con modelo esfera prueba 2

El experimento de no usar el algoritmo con el umbral máximo es útil, puesto que los datos de entrada de la cámara no serán esferas perfectas, si no objetos reales de los que solo tendremos los puntos de una de las visiones, y no nubes de puntos completas.

He generado una nube de puntos aleatoria con un plano rodeado de puntos, para probar en esta nube de puntos distintos modelos geométricos de RANSAC, un plano, una línea, un círculo, un cilindro y una esfera.

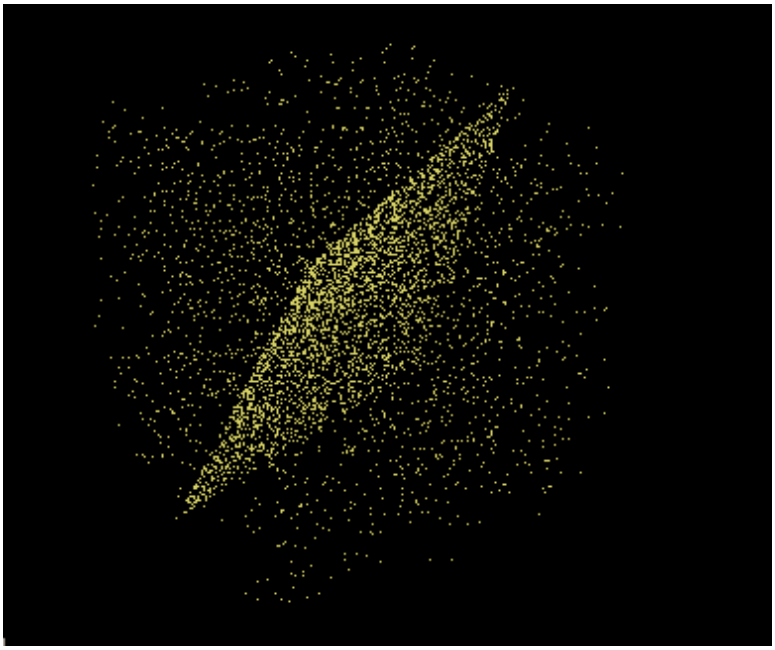


Figura4.15:Plano_amarillo

De los datos generales del algoritmo introduciremos:

Entrada:

- Datos de entrada: Plano_amarillo.
- Modelo: Plano, línea, circulo 2D, cilindro y esfera.
- **k** : 1000.
- **t**: 3,36505e-317 es el valor mínimo, el cual indica que los datos de entrada se ajustan al 100% al modelo.

Los resultados con el valor t mínimo son los siguientes:

- Plano: 1291 son los puntos que forman el plano. La nube de puntos se representa en la Figura4.16.

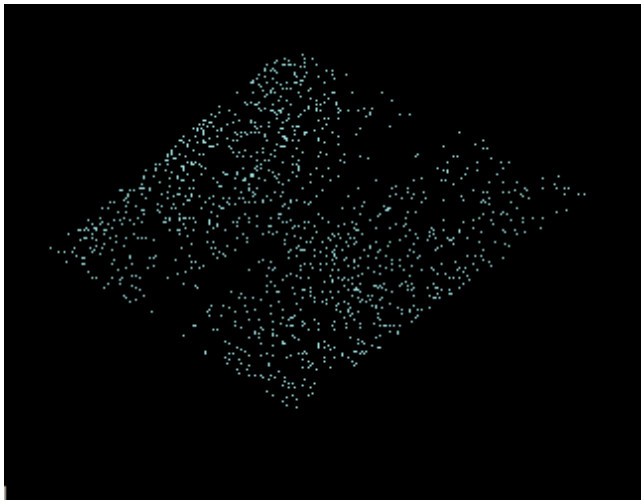


Figura 4.16: Plano_amarillo modelo plano prueba 1

- Línea: No hay inliers.
- Circulo 2D: Solo tres puntos pertenecen a inliers.
- Cilindro: Solo cinco puntos pertenecen a inliers.
- Esfera: No hay inliers que pertenezcan al modelo esfera en la nube de puntos de entrada.

Probamos a aumentar el umbral, t=0,05, y probar con los modelos más significativos para esta nube de puntos, el plano y la línea.

- Plano: 2631 son los puntos que forman ahora el plano, es un plano mas completo, con los bordes mas definidos, se representa en la Figura4.17.

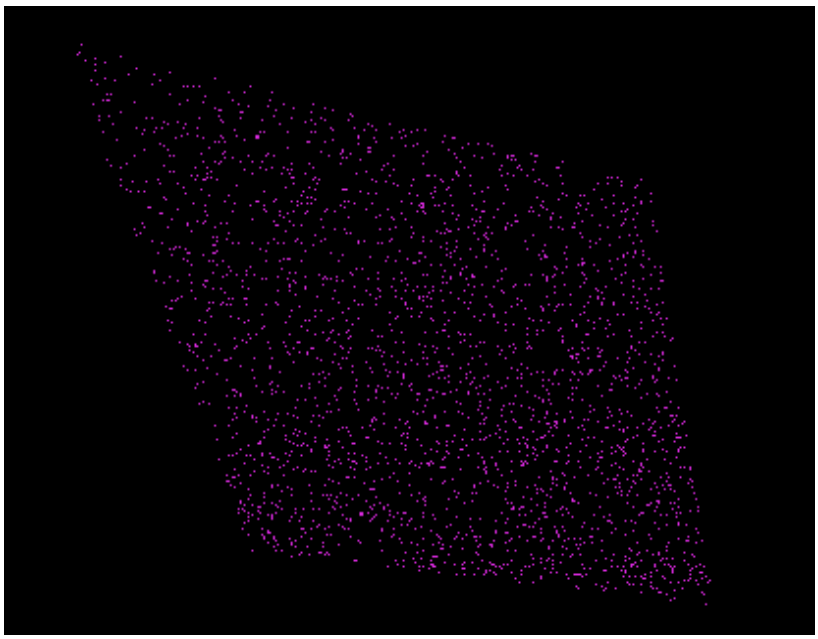


Figura 4.17: Plano_amarillo modelo plano prueba 2

- Línea: La diagonal de 186 puntos del plano es la línea que nos da como resultado el algoritmo con un umbral del 0,05 y un modelo de una línea.

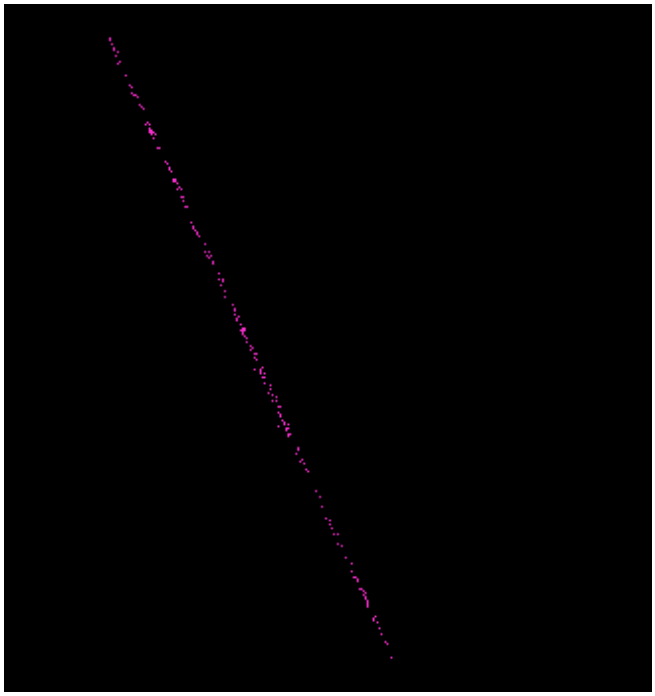


Figura 4.18: Plano_amarillo modelo línea

4.4. Normales

La normal de un punto se considera una característica local, porque para calcular la normal de un punto se basa en la información de los vecinos de dicho punto. Para buscar los vecinos más próximos de forma rápida y eficiente es muy común utilizar estructuras del tipo árbol. Dentro de este tipo de estructuras se encuentran los k-d Tree [13,14] (árboles de k dimensiones) esta será la estructura utilizada en el presente proyecto para acelerar la búsqueda de normales. Como condición para evaluar las normales en función de los vecinos se impone un radio alrededor del punto seleccionado.

El algoritmo de búsqueda k-d Tree (Bentley, 1975; Friedman, 1977) [15,16] es un algoritmo, de tipo árbol, que sirve para organizar estructuras de datos y así buscar en ellos de forma mas eficiente. Están compuestos por una serie de nodos y de ramas que acumulan información. La manera de trabajar es la siguiente, desde un conjunto de datos,k-d Tree busca el hiperplano (en el caso de nuestra nube de puntos en 3D, un plano) que corte el conjunto principal en dos subconjuntos para repetir el proceso con los mismos. Se busca un árbol equilibrado, que el prototipo (en este caso los vecinos) tenga la misma probabilidad de estar a un lado o a otro del plano.

La Figura4.19 es una sencilla imagen en la que se ve gráficamente como en cuatro pasos el algoritmo localiza los puntos.

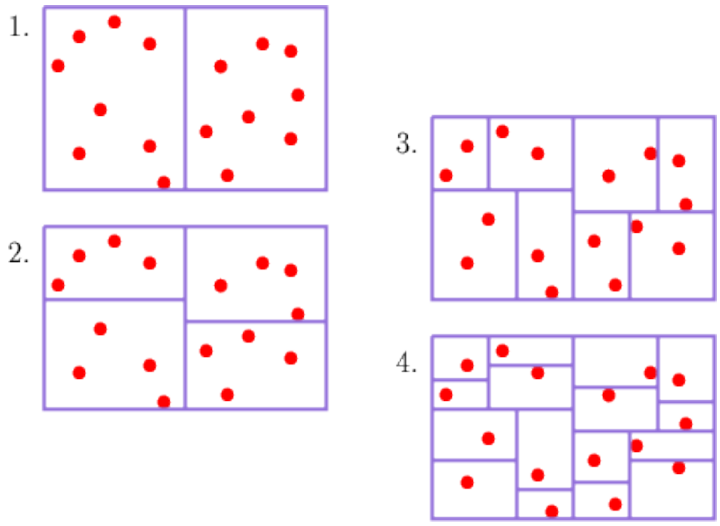


Figura 4.19: Ejemplo k-d Tree

En la Figura4.20 se puede observar una búsqueda de k-d Tree en tres dimensiones.

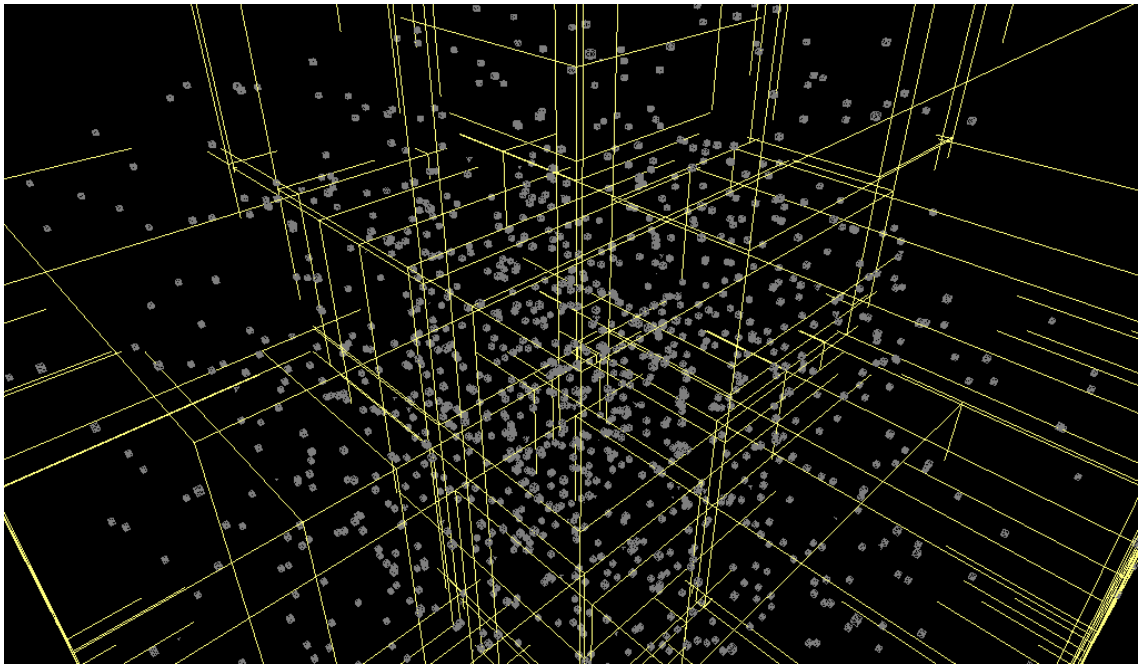


Figura 4.20: Segundo Ejemplo k-d Tree

Para trabajar con normales se usará, de la librería PCL, las clases “NormalEstimation” y “search/kdtree”. Como en este caso se trabaja con el método k-d Tree se creara un objeto de la clase “search::kdtree”, que será la entrada a la función “setSearchMethod” de la clase “NormalEstimation” y se crea un objeto de esta ultima que servirá para:

- Introducir la nube de puntos.
- Determinar el método de búsqueda.
- Indicar el radio de ejecución (m).
- Indicar la nube de salida donde se guardarán los datos de las normales.

Es necesario calcular las normales de cada posible objeto porque el algoritmo RANSAC necesita la nube de puntos original y las normales de la misma.

En las pruebas realizadas con nubes de puntos generadas por ordenador, el margen en el que se estudian los vecinos, marcado por una circunferencia de radio configurable, no es significativo siempre que el radio no sea mayor al tamaño total de la nube.

El archivo que contiene las normales se puede guardar en formato .pcd, formato explicado en la Sección3.2. En el caso de guardar normales en el nombre de cada dimensión encontraremos una imagen como la de la siguiente figura:

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
```

```
FIELDS normal_x normal_y normal_z curvature
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 33048
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 33048
DATA ascii
0.23376395 -0.94906718 -0.21124835 0.17173269
0.19424535 -0.91565973 -0.35190332 0.10003348
0.10704342 -0.92636448 -0.3610962 0.062751234
-0.028241711 -0.90709144 -0.419985 0.034508057
-0.011478485 -0.90019375 -0.43533835 0.034427796
0.0057939235 -0.88363552 -0.46813959 0.031662427
0.12159753 -0.88141942 -0.45641401 0.051995587
.
.
.
```

Figura 4.21: Archivo .pcd de normales

4.5. Modelos Planos

Después de adecuar la imagen, como se muestra en la Sección4.2, en un prisma en el que sólo deberíamos encontrar el objeto a estudiar en un plano, el siguiente paso es separar el objeto del plano que contiene su base. Para ello se utilizará el modelo de un plano (PERPENDICULAR_PLANE) y se estudiará su correspondencia en la nube de puntos mediante el algoritmo RANSAC. En las Figuras 4.22 y 4.23 se muestra el modelo de un plano desde dos vistas distintas.

Figura 4.22: Modelo plano vista 1

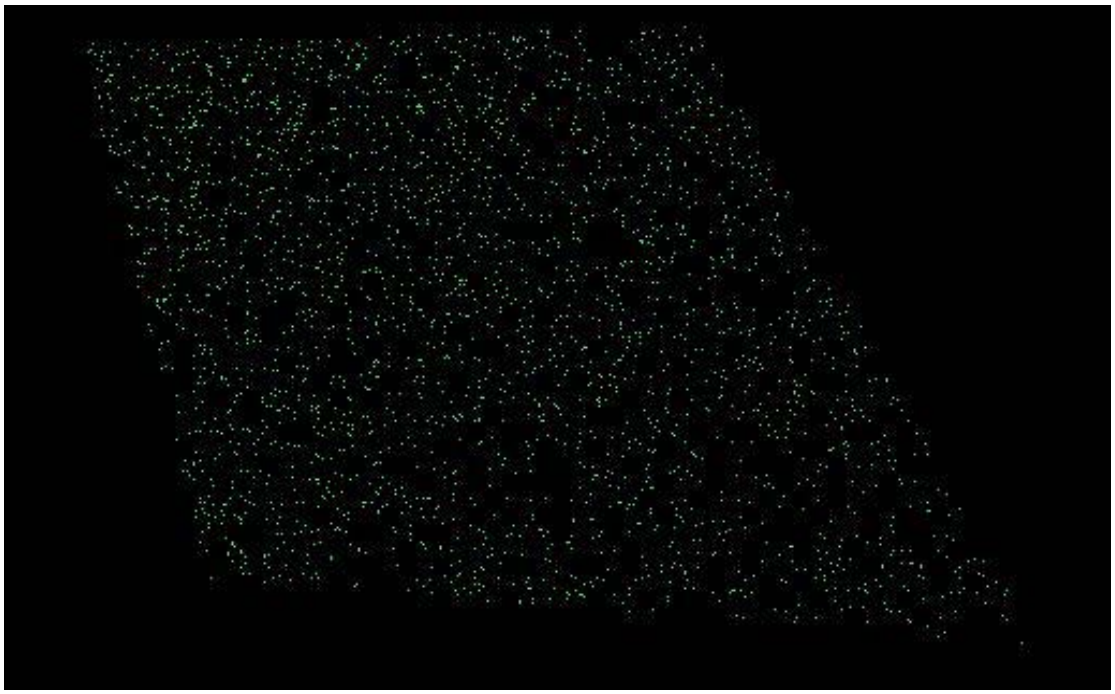


Figura 4.23: Modelo plano vista 2

Para ello como se explica en la Sección4.3, las entradas serán la nube de puntos original, sus correspondientes normales,y el modelo del plano. Los parámetros que podemos variar son tres:

- Umbral.
- Iteraciones máximas.
- Ángulo de la normal.

A continuación probaremos a modificar los tres parámetros, el máximo número de iteraciones que realizara el algoritmo para hallar la correspondencia con el modelo, el valor del umbral, el cual indica lo restrictivo que es el algoritmo para encontrar inliers en la nube de puntos de entrada, y en ángulo de la normal.

Figura 4.24: Habitación con silla a color

La imagen utilizada, la correspondiente a las Figuras 4.24 y 4.25, es una captura real de una estancia con una silla, pared, hueco de una puerta y suelo. La imagen es buena para probar puesto que tiene distintos planos a distintas alturas y la superficie del suelo no es grande en el contexto de la foto. La imagen tiene 307200 puntos.

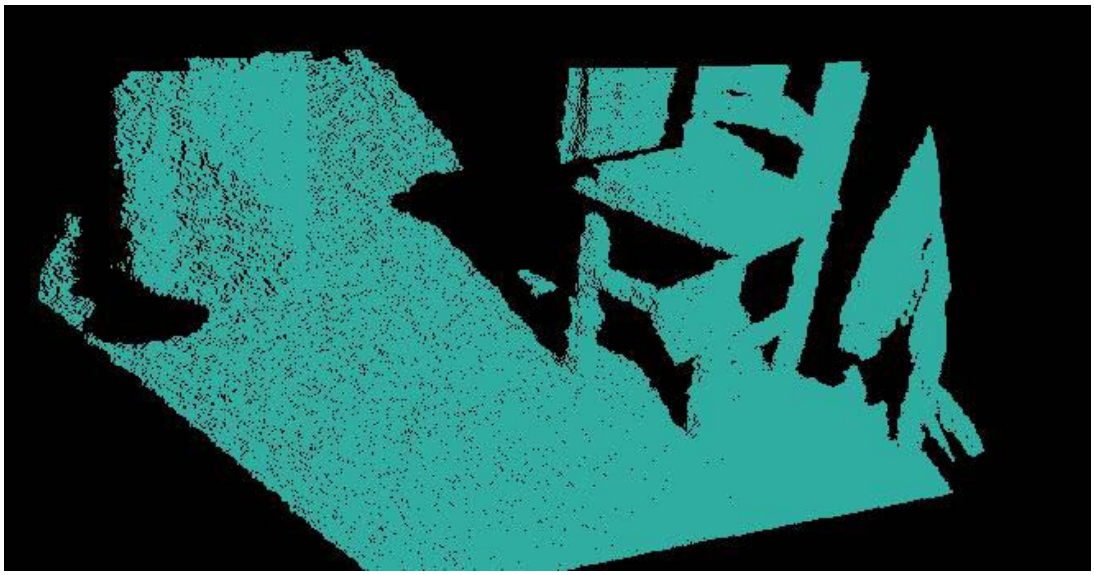


Figura 4.25: Habitación con silla

En la Figura4.26 se pueden ver las normales de la habitación.

Figura 4.26: Habitación con silla normales

La primera prueba en la búsqueda del modelo del plano en la imagen será con 1 iteración y 0,01 de umbral y 90º en el ángulo de la normal. Sabiendo que cuando menor es el umbral mas restrictiva es la búsqueda.

El algoritmo da como respuesta una nube de 8443 puntos sin un orden concreto, es un resultado insuficiente.

En la segunda prueba se aumenta el número de iteraciones a 10 para buscar un resultado.

Figura 4.27: Suelo de habitación con silla prueba 2

El algoritmo da como respuesta 117952 puntos. Es una imagen del suelo aceptable pero solo incluye un 38% de los puntos iniciales. Es recomendable seguir variando parámetros hasta conseguir el resultado mas optimo.

En la tercera prueba se aumenta a 100 las iteraciones siendo aún 0,01 el umbral.

Figura 4.28: Suelo de habitación con silla prueba 3

El resultado son 140678 puntos, la imagen corresponde al suelo de la habitación y el numero de puntos es un 46% del total.

En las siguientes pruebas se disminuirá el ángulo para no ser tan restrictivo, este parámetro no afecta en el resultado.

En conclusión usaremos un umbral de 0,01, el numero de iteraciones máxima no es relevante por encima de los 100 al igual que el ángulo de las normales.

4.6. Reconocimiento de prismas

Para el proceso de reconocimiento de la forma en 3D prisma, se utilizará un procedimiento basado en estudiar los planos que forman dicho prisma. El modelo RANSAC utilizado será PERPENDICULAR_PLANE. Dicho modelo determina un plano.

Según la Real Academia Española, un prisma es Cuerpo limitado por dos polígonos planos, paralelos e iguales que se llaman bases, y por tantos paralelogramos cuantos lados tenga cada base. Los prismas que dan objeto a este trabajo son los rectangulares, formados por seis caras rectangulares, pero con la cámara solo veremos tres caras que son las que definirán el prisma. La Figura4.29 corresponde a un prisma rectangular estándar.

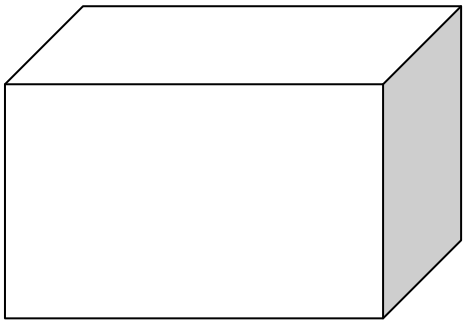


Figura 4.29: Prisma rectangular

Los objetos que en el mundo real más representan al prisma rectangular son las cajas. Por lo que será el objeto elegido para realizar pruebas. En la Figura4.30 se pueden ver nubes de puntos capturadas con la Kinect correspondientes a cajas.

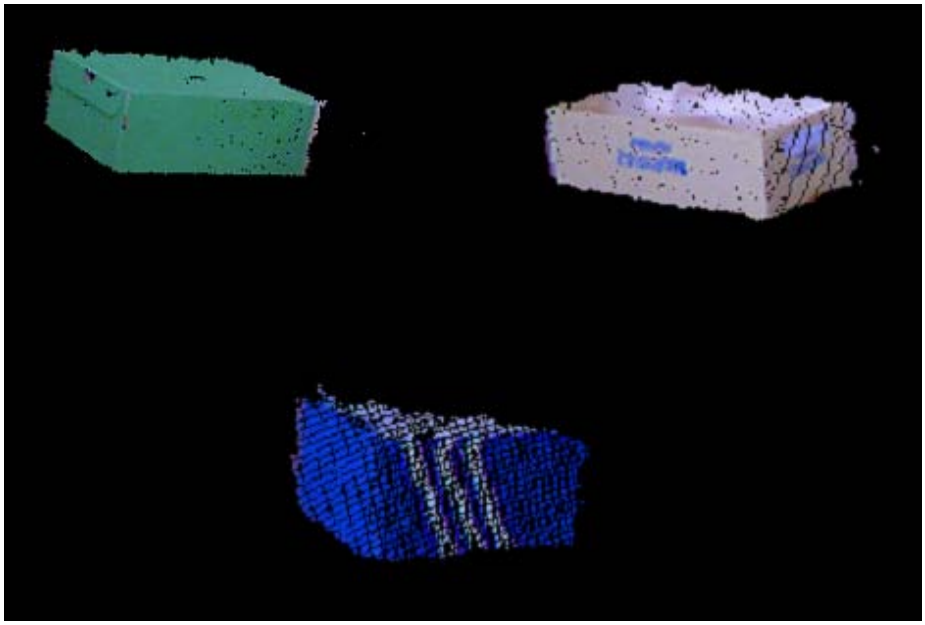


Figura 4.30: Caja verde, caja azul y caja amarilla

Para encontrar el mejor plano de la nube de puntos cada vez que ponemos en marcha el algoritmo RANSAC (una desventaja del algoritmo es que solo puede analizar con un modelo cada vez) Usaremos las siguientes restricciones:

- Umbral 0,01
- Iteraciones máximas 1000
- Ángulo de la normal 18º

Como objeto principal se utilizara la caja azul de dimensiones (33x11x19cm) y 3911 puntos.

Con estas restricciones obtenemos el plano de la Figura4.31de 1904 puntos, el algoritmo buscará el plano mayor de la imagen, a este plano de mayor extensión se le denominará primer plano.En el caso de la caja azul, el primer plano ocupa el 48% del prisma.

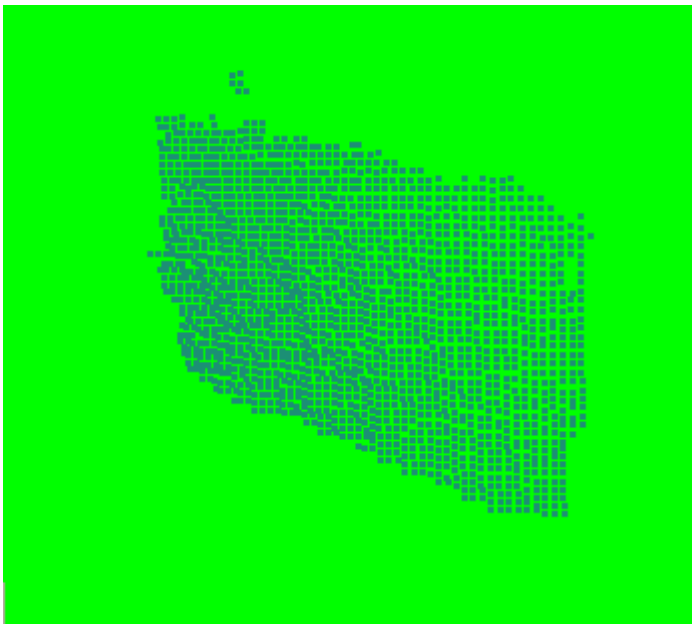


Figura 4.31: Caja azul plano 1

También podemos quedarnos con la nube sin ese primer plano (el más grande de los tres), se muestra en la Figura4.32.

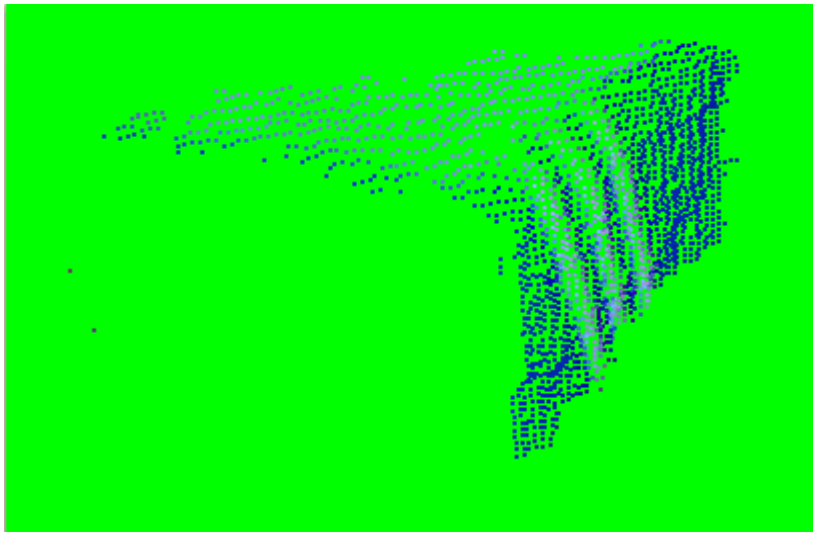


Figura 4.32: Caja azul sin plano 1

El procedimiento será el esquematizado en la Sección 4.1. Se analizará la figura inicial de tal forma que el algoritmo nos devolverá el primer plano y se calculará el porcentaje que ocupa del prisma total.A continuación teniendo como dato de entrada la nube de puntos sin el primer plano, Figura 4.31, se aplicará al algoritmo RANSAC el modelo plano hasta obtener el segundo plano más significativo el cual ocupará un porcentaje de la imagen. En la tercera búsqueda de plano se obtendrá la tercera cara visible del prisma y como en el caso de los planos anteriores se calculará el porcentaje de puntos que ocupa del total del objeto. Estos porcentajes serán utilizados para determinar si el objeto del estudio es un prisma.

El segundo plano mas extenso es el mostrado en la Figura4.33, ocupa un 38% del total.

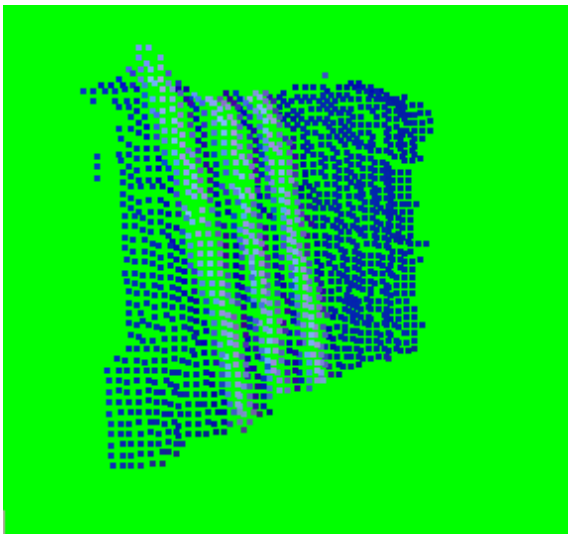


Figura 4.33: Caja azul plano 2

El tercer y último plano es el mostrado en la Figura4.34, ocupa un 12% del total.

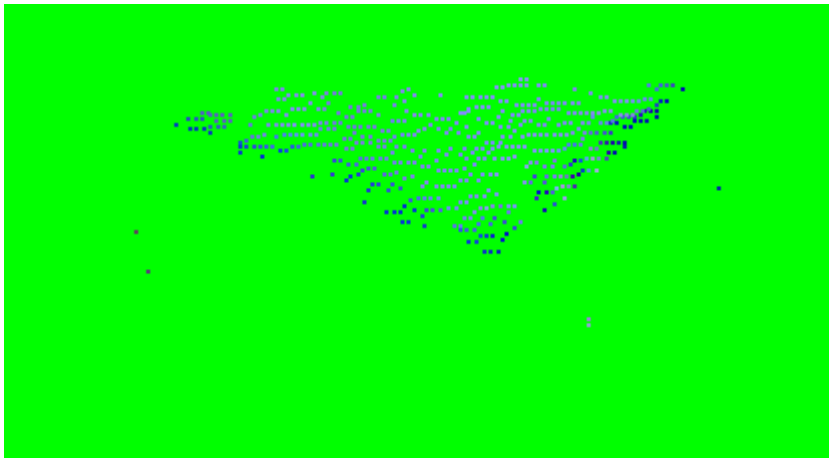


Figura 4.34: Caja azul plano 3

Para las cajas amarilla, azul y verde los resultados son los que se indican en la tabla de la Figura4.35.

	Verde	Azul	Amarilla
Plano 1	42%	48%	55%
Plano 2	35%	38%	30%
Plano 3	21%	12%	13%

Figura 4.35: Tabla de planos

Según los resultados anteriores se puede definir que en un prisma rectangular los tres planos visibles ocuparan:

- Plano 1 entre un 40% y un 60%.
- Plano 2 entre un 25% y un 40%.
- Plano 3 entre un 10% y un 25%.

Si una figura esta compuesta por tres planos y estos tienen las anteriores dimensiones la nube de puntos se considerará un prisma rectangular.

4.7. Reconocimiento de esferas

Para el proceso de reconocimiento de la forma en 3D esfera, se utilizara el algoritmo RANSAC con el modelo SPHERE, en la figura siguiente podemos ver un modelo de esfera.

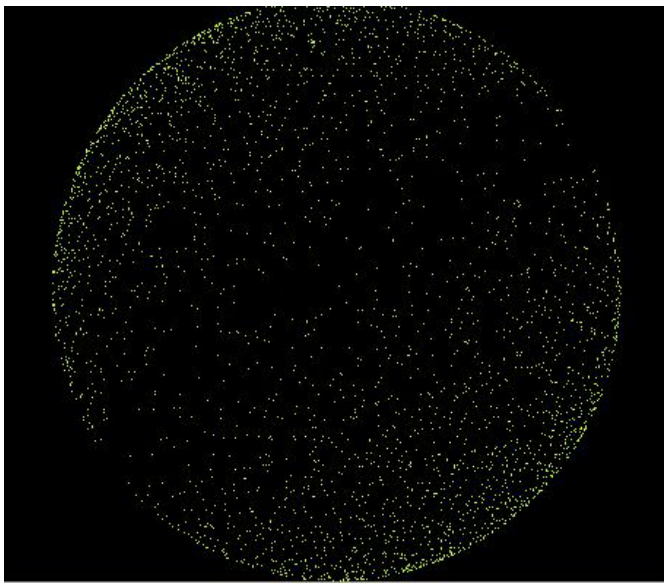


Figura 4.36: Modelo esfera

Para ello como se explica en la Sección4.3, las entradas serán la nube de puntos original, sus correspondientes normales,y el modelo del plano. Los parámetros que podemos variar son cuatro:

- Umbral.
- Iteraciones máximas.
- Ángulo de la normal.
- El valor mínimo y máximo del radio permitido.

A continuación probaremos a modificar los cuatro parámetros, el máximo número de iteraciones que realizara el algoritmo para hallar la correspondencia con el modelo, el valor del umbral, el cual indica lo restrictivo que es el algoritmo para encontrar inliers en la nube de puntos de entrada, el ángulo de la normal y el rango del radio.

Para las pruebas se trabajará con la nube de puntos correspondiente a una pelota, la imagen de la pelota se puede ver en las Figuras 4.37 y 4.38. La nube de puntos es una imagen real tomada con la cámara, y adaptada según la Sección4.2, el suelo se ha

eliminado según las conclusiones de la Sección 4.5. La pelota tiene un radio de 10 cm y se compone de 4726 puntos.

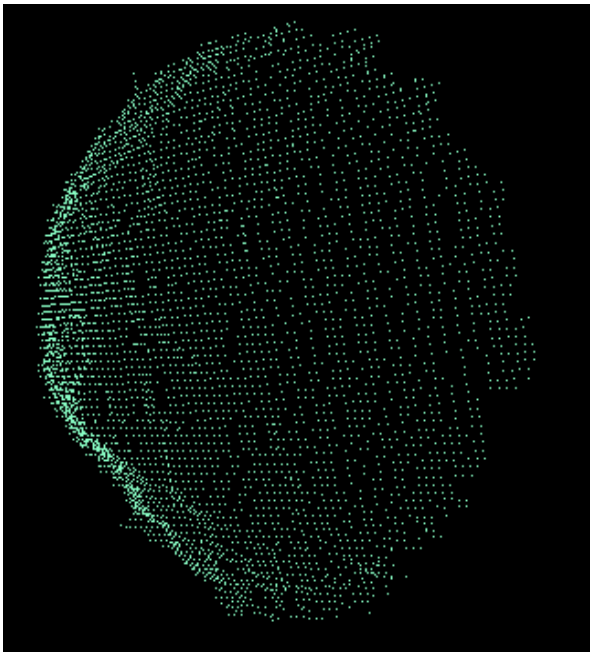


Figura 4.37: Captura pelota

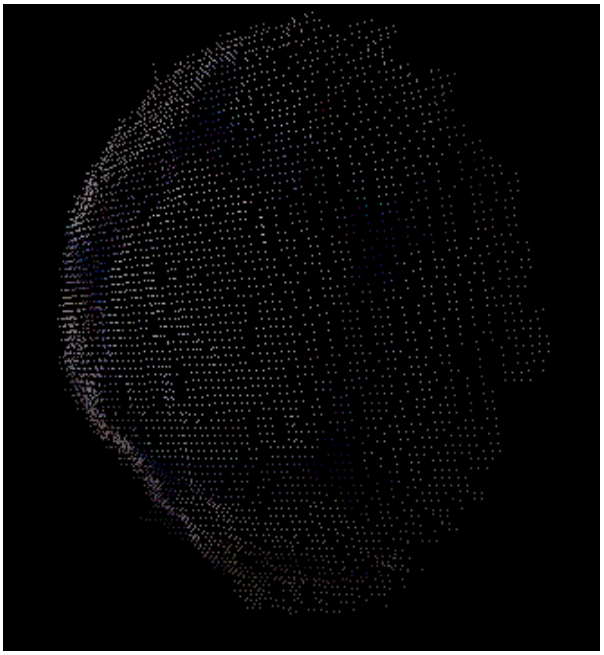


Figura 4.38: Captura pelota color

Empezaremos a trabajar con un umbral de 0,1, el numero de iteraciones máxima sera 100, un ángulo en las normales de 45º y un rango entre 0 y 20 cm en el radio. Estas características de partida son las conclusiones para un uso adecuado del modelo plano en el algoritmo RANSAC.

Con estas condiciones el algoritmo nos devuelve el 100% de los puntos como inliers. Conviene ser mas restrictivos y se aplicará un umbral de 0,01.

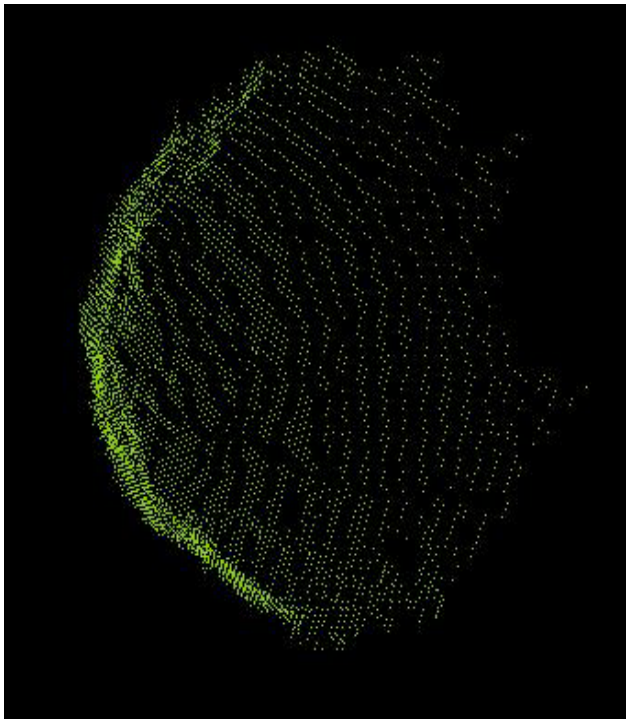


Figura 4.39: Resultado reconocimiento esfera pelota

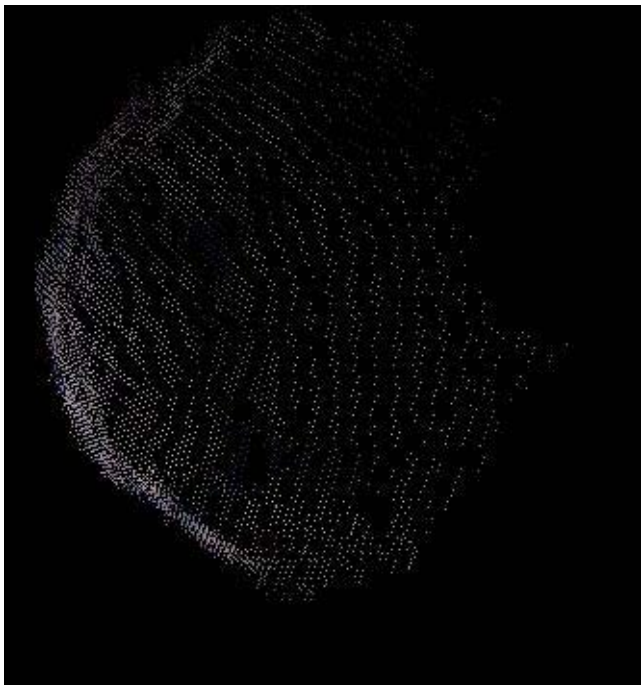


Figura 4.40: Resultado reconocimiento esfera pelota color

El resultado es muy bueno, la nube tiene 4686 un 99,15% de puntos de la imagen inicial, y con el umbral mas bajo nos aseguramos mas fidelidad al modelo. Las Figuras 4.39 y 4.40 corresponden al resultado del reconocimiento de esferas con una pelota.

4.8. Reconocimiento de cilindros

Para el proceso de reconocimiento de la forma en 3D cilindro, se utilizara el algoritmo RANSAC con el modelo CYLINDER como el de la Figura4.41.

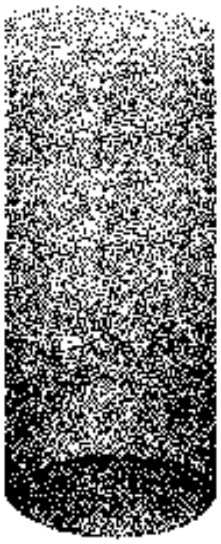


Figura 4.41: Modelo cilindro

Para ello como se explica en la Sección4.3, las entradas serán la nube de puntos original, sus correspondientes normales y el modelo del plano. Los parámetros que podemos variar son cuatro:

- Umbral
- Iteraciones máximas
- Ángulo de la normal
- El valor mínimo y máximo del radio permitido

A continuación probaremos a modificar los cuatro parámetros, el máximo número de iteraciones que realizara el algoritmo para hallar la correspondencia con el modelo, el valor del umbral, el cual indica lo restrictivo que es el algoritmo para encontrar inliers en la nube de puntos de entrada, el ángulo de la normal y el rango del radio.

Para las pruebas se trabajará con la nube de puntos correspondiente a una lata, representada en la Figura4.42 a color y en la Figura4.43. La nube de puntos es una imagen real tomada con la cámara, y adaptada según la Sección4.2, el suelo se ha eliminado según las conclusiones de la Sección4.5. La lata tiene un radio de 10,5cm y se compone de 2143 puntos.

Figura 4.42: Lata blanca a color

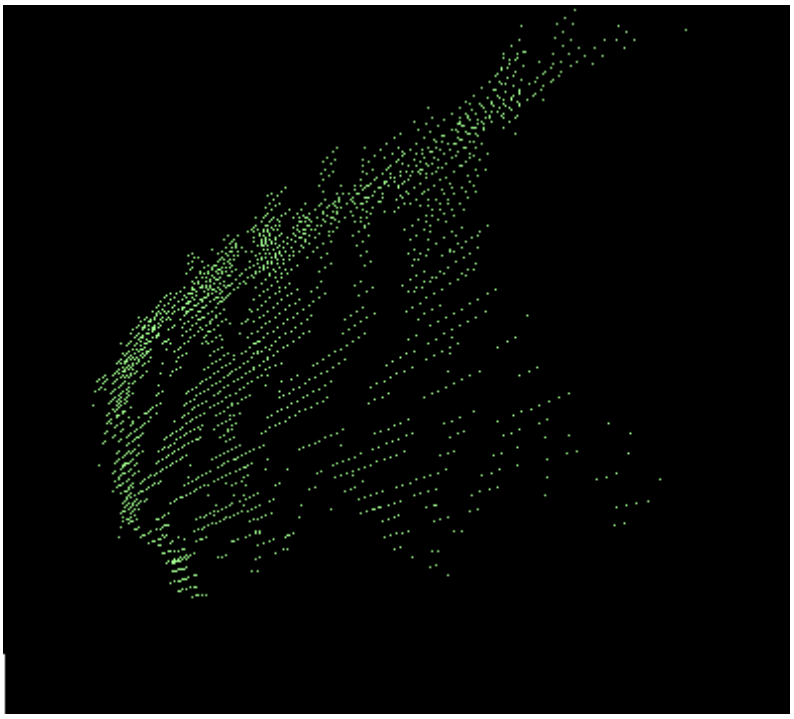


Figura 4.43: Lata blanca

Empezaremos a trabajar con un umbral de 0,01, el número de iteraciones máxima será 100, un ángulo en las normales de 45º y un rango entre 0 y 20 cm en el radio. Estas características de partida han sido correctas en el modelo de la esfera.

Tras la primera prueba el algoritmo no devuelve ningún dato, el umbral es demasiado restrictivo, en las siguientes pruebas se aumenta el umbral obteniendo los siguientes resultados:

Umbral=0,03 → Número de puntos=56

Umbral=0,07 → Número de puntos=268

Umbral=0,1 → Número de puntos=537

Umbral=0,3 → Número de puntos=1696

Con 1696 puntos, un 79% de los datos iniciales, es suficiente para determinar a la vista la forma cilíndrica y es suficientemente restrictivo. En las Figuras 4.44 y 4.45 se pueden ver el resultado con 1696 puntos, en la Figura4.44 se ve el perfil a color y la Figura4.45 corresponde a una imagen frontal. Estos parámetros serán la referencia mas adelante para el proceso total de reconocimiento de formas, es decir en este momento sabemos que la nube de entrada es un cilindro pero este proceso se repetirá con formas como prismas o esferas y si no lo hacemos suficientemente restrictivo cualquier objeto será un cilindro.

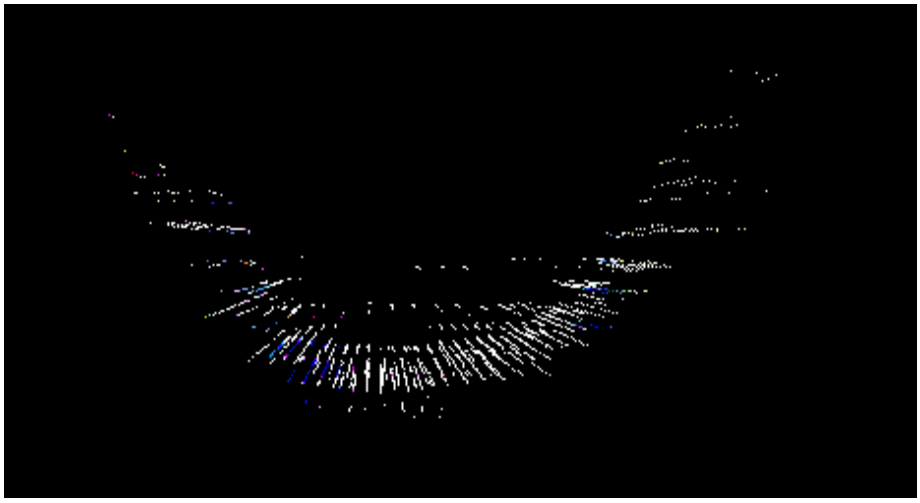


Figura 4.44: Resultado reconocimiento cilindro lata blanca color

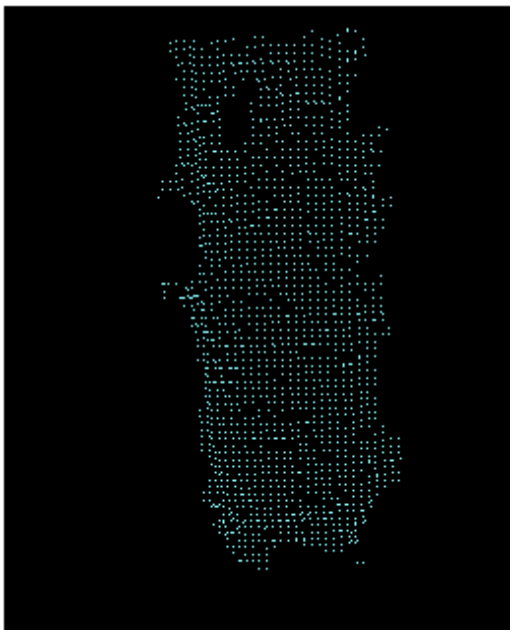


Figura 4.45: Resultado reconocimiento cilindro lata blanca

5. Resultados experimentales

En este capítulo se engloban pruebas experimentales del programa completo en el que se presentaran objetos a la cámara y se diferenciaran esferas, cilindros y prismas. Se presentan cuatro pruebas. Una prueba con un prisma el cual no fue utilizado en la Sección4.6 que se carazteriza el reconocimiento de prismas. Se detallan dos pruebas con cilindros de distintas caracteristicas y una pruena con una esfera.

Los parámetros para el uso del algoritmo RANSAC , Sección4.3, serán inicialmente los probados en las Secciónes 4.5, 4.6, 4.7 y 4.8. Los parametros a modificar son:

- Umbral
- Iteraciones máximas
- Ángulo de la normal

Los parametros iniciales son:

	Umbral	Iteraciones	Angulo
Planos	0,01	100	X
Prismas	0,01	1000	18º
Esferas	0,01	100	45º
Cilindros	0,3	100	45º

Figura 5.1: Parámetros iniciales reconocimiento de formas

Es importante evaluar el coste computacional del proceso de reconocimiento de formas. Para ello se van a evaluar, para las distintas pruebas, los tiempos de ejecución. Para conseguir un resultado más fiable se realizaran cinco tomas de tiempos para cada prueba y se tendrá en cuenta la media de los resultados.

5.1. Prueba 1 - Prisma

La primera prueba se realizará con una caja de (33x10x24cm)en una habitación como la mostrada en la Figura5.2.

Figura 5.2: Habitación prueba 1 prisma

El programa nos devuelve que la nube de puntos corresponde a un prisma, la correspondiente a la Figura5.3. La nube de puntos recortada son 15240:

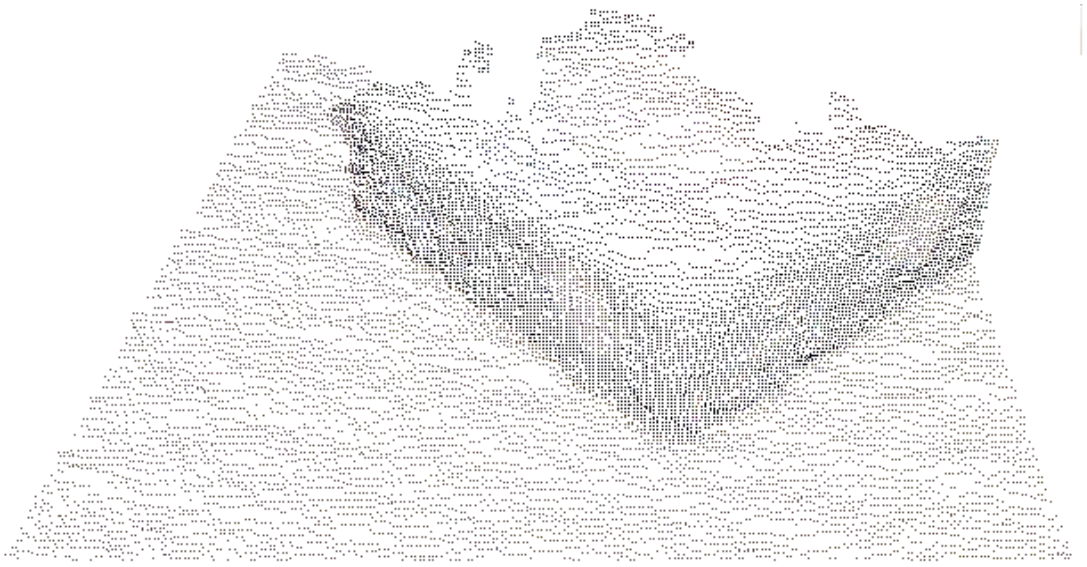


Figura 5.3: Imagen recortada prueba 1 prisma

El programa devuelve la siguiente nube, Figura5.4, de 7508 puntos correspondiente al prisma:

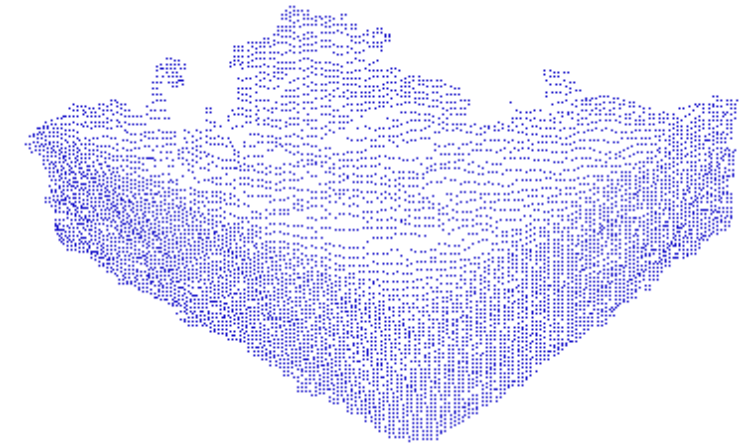


Figura 5.4: Prisma resultado prueba 1 prisma

El resultado es bueno, se observan los puntos correspondientes a la caja.

Para la primera prueba los tiempos (en milisegundos) tomados son los que se observan en la Figura 5.5:

T1	1784,22
T2	1799,95
T3	1798,04
T4	1785,58
T5	1784,64

Figura 5.5: Tabla tiempos prueba 1

La media de los tiempos es de 1790,48 milisegundos.

Prueba 2 - Cilindro

La segunda prueba se realizará con una de botella de 2,2dm3 y un diámetro de 10,5cm, en una habitación con más objetos, la habitación queda representada en la Figura 5.6.

Figura 5.6: Habitación prueba 2 cilindro

El programa devuelve la nube de puntos correspondiente a la botella indicando que es un cilindro. La nube solo tiene 1919 puntos,Figura 5.7, es suficiente para que el programa lo distinga como cilindro pero es muy pobre para una representación.

Figura 5.7: Cilindro resultado prueba 2 cilindro

El objeto de solo 1919 puntos es escaso para representar el objeto. Para evitar nubes de puntos pequeñas solo se aceptaran nubes de puntos de esferas o cilindros por encima de 1000 puntos.

Para la segunda prueba los tiempos (en milisegundos) tomados son los que aparecen en la Figura 5.8.

T1	1213,06
T2	1148,92
T3	1159,85
T4	1151,49
T5	1154,58

Figura 5.8: Tabla tiempos prueba 2

La media de los tiempos es de 1165,58milisegundos.

5.2. Prueba 3 - Cilindro

La tercera prueba será un cilindro y los valores iniciales de los parametros seran los mismos que en la segunda prueba, el objeto será un ambientador de 4,5cm de radio y 20cm de alto en una habitación con mas objetos.

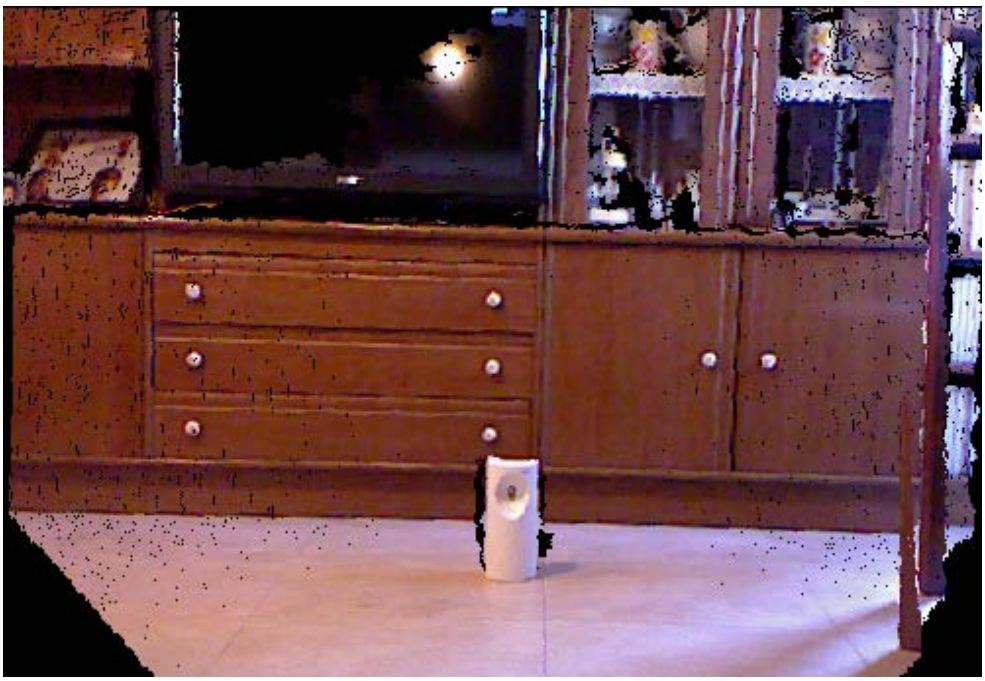


Figura 5.9: Habitación prueba 3 cilindro

El programa devuelve una nube de puntos de un cilindro, se representa en la Figura 5.10. La nube de puntos tiene 1692 puntos, son pocos puntos para identificar el objeto, el resultado es equivalente q al de la prueba número dos con la botella.

Figura 5.10: Cilindro primer resultado prueba 3 cilindro

Para intentar mejorar el resultado, se variará el umbral. El valor inicial es 0,3, disminuyendo el umbral, es decir aumentando el porcentaje de coincidencia con el modelo, el número de puntos es menor. Con umbral 0,1 obtendremos 418 puntos.

Aumentando el umbral a 0,4 la nube de puntos aumenta a 2309, el resultado es mucho mejor y como se puede observar en la Figura 5.11 se distingue mejor el objeto. Si seguimos aumentando el umbral el resultado empeora.

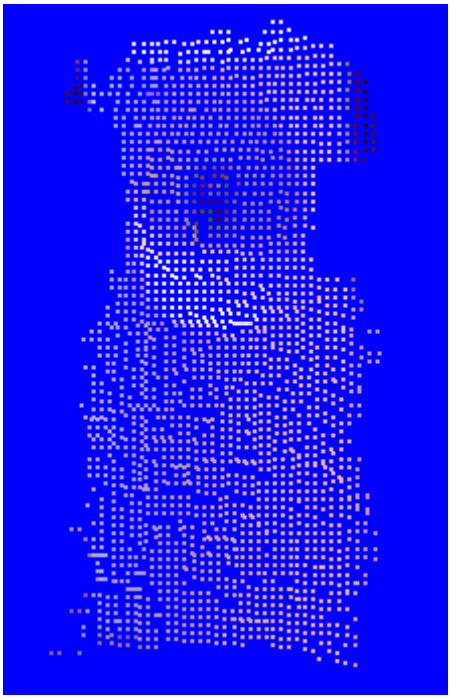


Figura 5.11: Cilindro segundo resultado prueba 3 cilindro

En la tercera prueba los tiempos (en milisegundos) tomados son los correspondientes a la Figura 5.12.

T1	1324,73
T2	1301,78
T3	1303,76
T4	1313,55
T5	1310,29

Figura 5.12: Tabla tiempos prueba 3

La media de los tiempos es de 1310,82milisegundos.

Prueba 4 - Esfera

La cuarta prueba corresponde a una habitación, Figura 5.13, en la que en el suelo se encuentra una pelota de fútbol de 20cm de diámetro.

Figura 5.13: Habitación prueba 4 esfera

El programa distingue una esfera y devuelve una nube de 4168 puntos, que se muestra en la Figura 5.15 y en la Figura 5.14 a color.

Figura 5.14: Esfera a color resultado prueba 4 esfera

Figura 5.15: Esfera resultado prueba 4 esfera

La imagen es suficientemente buena como para distinguir la pelota.

Los tiempos medidos para el reconocimiento de la esfera en milisegundos) tomados son los mostrados en la Figura 5.16.

T1	1546,22
T2	1538,6
T3	1541,87
T4	1532,93

Figura 5.16: Tabla tiempos prueba 4

La media de los tiempos obtenidos es de 1537,78 milisegundos.

5.3. Prueba 5 – Cilindro

En la quinta prueba se probará la captura de la cámara con objetos transparentes. En primer lugar se probará la botella de la prueba 2, detallada en la Sección 5.2. En este caso la botella está vacía como se muestra en la Figura 5.17.

Figura 5.17: prueba 5 cilindro

La imagen recortada, Figura 5.18, muestra que la imagen no detecta el objeto. Por supuesto el software no detecta ninguna forma.

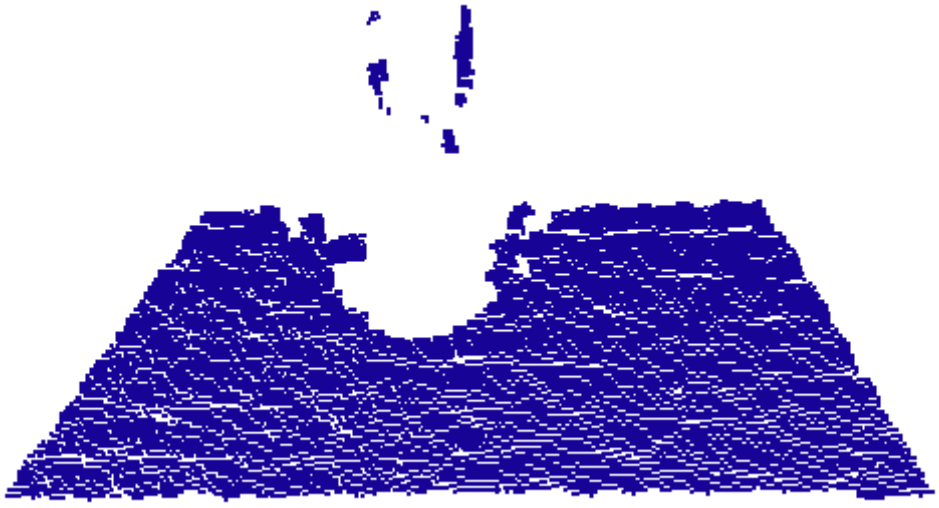


Figura 5.18: Imagen recortada prueba 5 cilindro

En este mismo experimento se probará con otra forma cilíndrica, una tarrina de CD vacía. La captura de la imagen, que corresponde a la Figura 5.19, muestra que no se detecta objeto.

Figura 5.19: segunda prueba 5 cilindro

En la Figura 5.20 se muestra el cilindro con un papel para que sea opaco, y de este modo la cámara lo detecte.

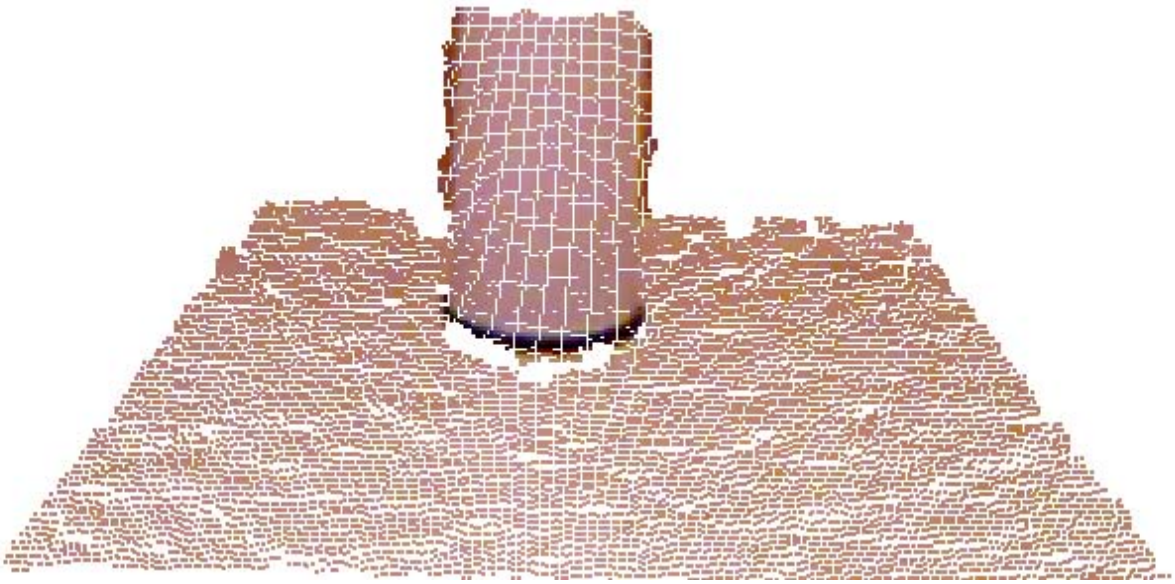


Figura 5.20: Imagen recortada segunda prueba 5 cilindro

El programa detecta una esfera en vez de un cilindro, la causa posible es que el objeto no tiene lado superior. Para comprobarlo se le coloca una tapa a la tarrina de CD opaca. En esta ocasión el programa detecta un cilindro de 3568 puntos la imagen correspondiente es la Figura 5.21.

Figura 5.21: Resultado segunda prueba 5 cilindro con tapa

En la quinta prueba correspondiente al reconocimiento de un cilindro opaco con tapa los tiempos obtenidos (en milisegundos) son los correspondientes a la Figura 5.22.

T1	1419,32
T2	1457,85
T3	1460,19
T4	1455,28

Figura 5.22: Tabla tiempos prueba 5

La media de los tiempos obtenidos es de 1452,34 milisegundos.

5.4. Prueba 6 – Prisma

La sexta prueba se divide en dos partes. En la primera se pone a prueba al programa con un libro, es un prisma con la superficie brillante y las aristas no están totalmente definidas.

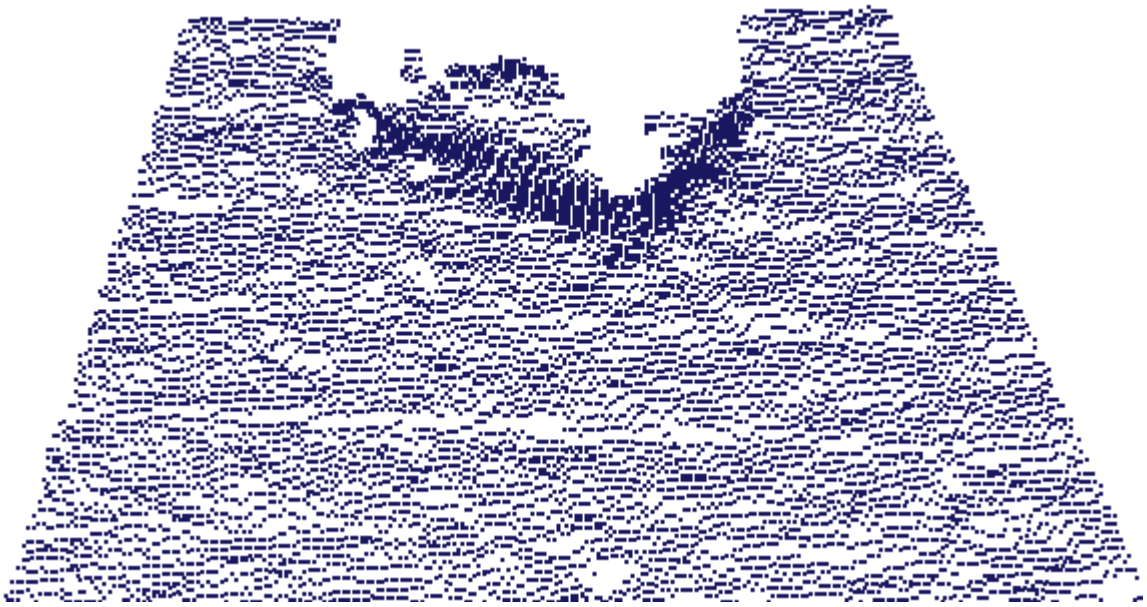


Figura 5.23: Imagen recortada prueba 6 prisma horizontal

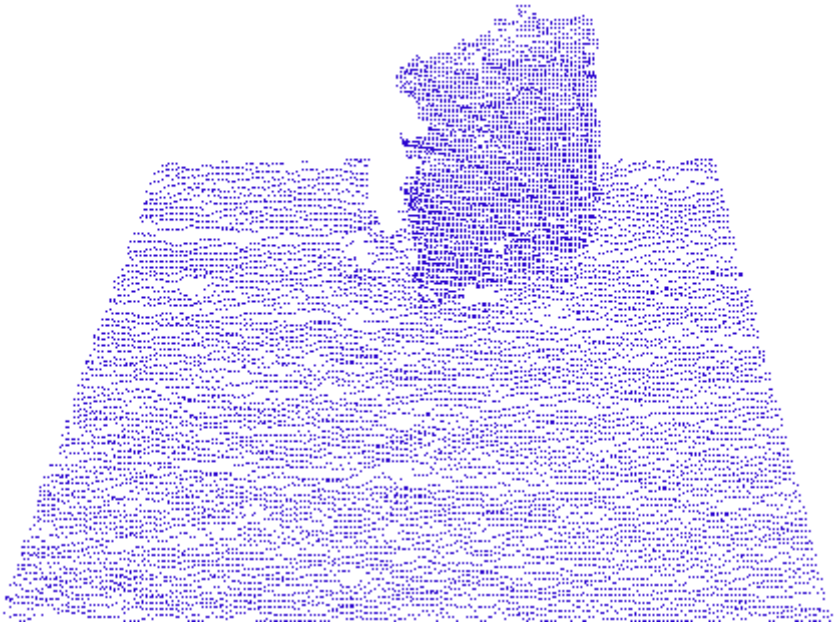


Figura 5.24: Imagen recortada prueba 6 prisma vertical

En las Figuras 5.23 y 5.24 se muestran dos capturas de imagen del libro. El programa no reconoce ninguna forma.

En la segunda parte de la prueba el libro tiene una superficie mate y tiene una forma mejor definida que en la Figura 5.23.

Figura 5.25: segunda prueba 6 prisma

Figura 5.26: resultado segunda prueba 6 prisma

Con este mismo objeto, correspondiente a las Figuras 5.25 y 5.26, se pone a prueba el software en una escena totalmente a oscuras.

Gracias a que la cámara se divide en dos sensores independientes, la cámara a color RGB y el sensor infrarrojos, es posible la captura sin luz, Figura 5.27. Esta opción puede ser útil en algunas aplicaciones en las que el color no sea determinante.

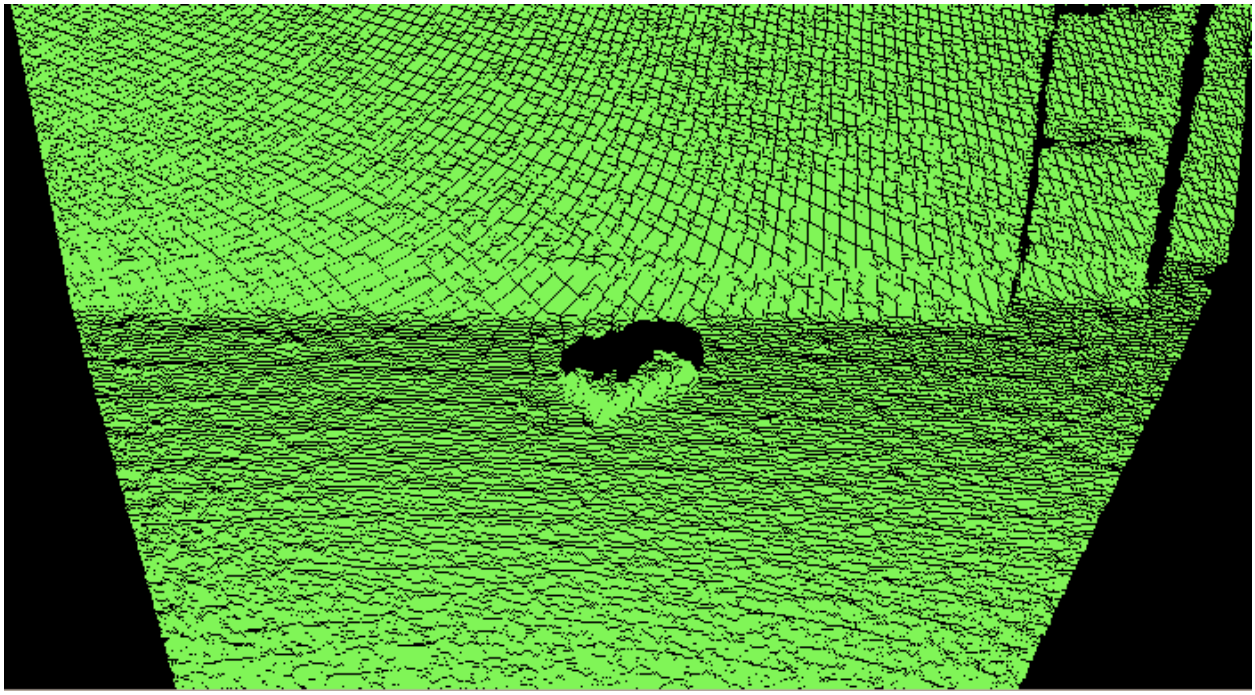


Figura 5.27: segunda prueba 6 prisma sin iluminación

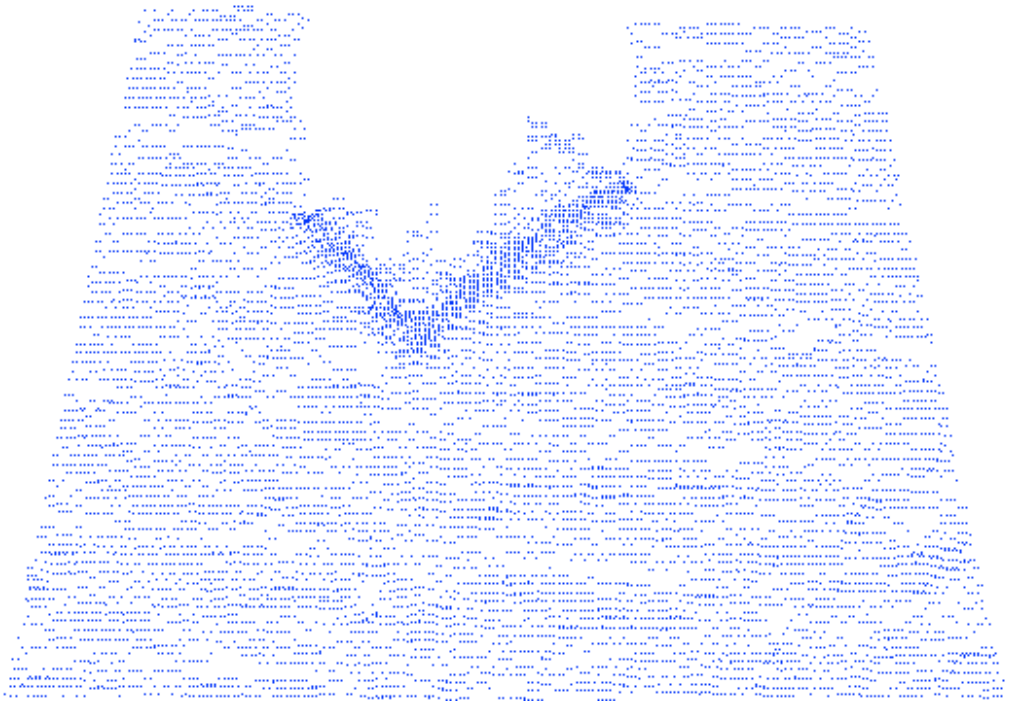


Figura 5.28: imagen recortada segunda prueba 6 prisma sin iluminación

El resultado de la prueba a oscuras, correspondiente a la Figura 5.28, es igual de válido que el de la escena iluminada, esto demuestra que no es necesaria la luz para el uso de la cámara y del software desarrollado en el presente trabajo.

En esta prueba se han tomado dos tiempos los correspondientes a la Figura 5.24, los cuales (en milisegundos) son los mostrados en la Figura 5.29.

T1	1021,1
T2	1006,73
T3	1006,92
T4	1018,56

Figura 5.29: Tabla 1 tiempos prueba 6

La media de los tiempos obtenidos es de 1012,69 milisegundos.

También se han tomado los tiempos (en milisegundos) de la escena sin iluminación y se muestran en la Figura 5.30.

T1	1003,79
T2	1005,7
T3	1005,58
T4	1005,71

Figura 5.30: Tabla 2 tiempos prueba 6

La media de los tiempos obtenidos es de 1007,22 milisegundos.

6. Conclusiones

El objetivo principal de este proyecto es el desarrollo de un software en C++ capaz de capturar imágenes en 3D en un PC y tratarlas para determinar modelos geometricos. Los objetivos del trabajo se han cumplido satisfactoriamente.

La captura de imágenes con la cámara Kinect para un PC, se ha realizado con éxito, permitiendo conocer los archivos que contienen nubes de puntos, .pcd, como tratar esos archivos para poder recortar imágenes, aplicar complejos algoritmos o simplemente visualizarlos desde distintos ángulos.

Los algoritmos usados para identificar las formas geométricas se han basado en métodos de búsqueda y modelos matemáticos RANSAC, este trabajo permite conocer el poder de dicho algoritmo para de una compleja y abundante nube de puntos simplificar a modelos geométricos. Tambien este trabajo a permitido el uso del metodo de busqueda k-d tree, aplicado para la busqueda de vecinos y con ello mejorar la descripción de normales en superficies.

Se han realizado pruebas experimentales con la camara Kinect, permitiendo familiarizarse con esta nueva tecnologia que apunta al futuro de la automatizacion y robotica. Tambien se propone esta nueva tecnologia en un tratamiento para personas con problemas en centros ocupacionales.

Las principales dificultades en un inicio fueron familiarizarse con el entorno de programacion de Ubuntu, un sistema operativo desconocido para mi, el control de la camara, y el descubrimiento de las librerias PCL en el lenguaje C++. En las pruebas se ha detectado problemas con objetos brillantes o transparentes.

Una parte interesante de las pruebas realizadas ha sido el cálculo del tiempo de ejecución del programa, la media de tiempos de todas las pruebas ha sido de 1325,28 milisegundos, o lo que es lo mismo 1,32 segundos. El tiempo es pequeño y no hay grandes diferencias entre unas pruebas y otras, ni entre unas formas y otras. Hay una pequeña variación de tiempos siendo más pequeños para objetos con menos puntos. Se puede destacar el tiempo obtenido durante la sexta prueba, en especial la escena sin iluminación, siendo un 24% más pequeño de la media y un 44% más bajo que el tiempo mayor.

Respecto a las posibles mejoras, sería interesante combinar las formas geométricas con el color de las mismas y ampliar la gama de modelos geométricos.

El presente trabajo de modelado de objetos en tres dimensiones puede tener varias aplicaciones en el mundo de la robótica y la automatización. En el mundo de la robotica la Kinect con el presente software se podría adaptar al robot MANFRED-2 de la UC3M para que el manipulador identificara los objetos que debe o no manipular. En

la automatización se podría aplicar en una cadena de producción para clasificar objetos. En la Sección 6.1se detalla otra aplicación en este caso en el campo de la medicina.

6.1. Aplicación

Una aplicación posible al programa resultante de la modelación de objetos en 3D esta dirigida a la terapia con personas en hospitales, centros de día, centros ocupacionales, etc.

La aplicación de las nuevas tecnologías en la terapia es un tema en constante desarrollo, se puede destacar el uso de videoconsolas (Wii, Kinect Xbox, Move de PlayStation) para pacientes con autismo, problemas de movilidad, problemas cardiacos, etc. Esta aplicación es una solución a la falta de presupuesto para personal cualificado o de material, ya que el coste es bajo y algunos centros ya cuentan con la camara Kinect y por supuesto con un ordenador.

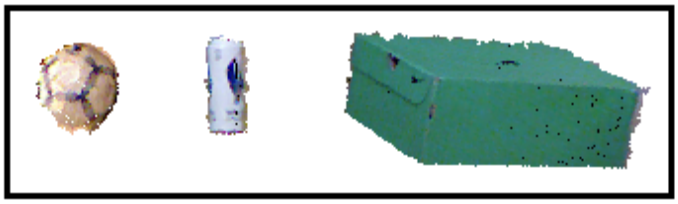


Figura 6.1: Ejemplo de serie en aplicación

La aplicación consiste en que al paciente se le plantee una serie compuesta con objetos, como el mostrado en la Figura6.1, y el paciente muestre a la cámara en el orden correcto los objetos. Esta aplicación permitiría el trabajo del paciente de manera autónoma, teniendo una relación camara usuario directa sin la necesidad de que un terapeuta este durante toda la terapia con el paciente. El algoritmo de reconocimiento de formas al estar desarrollado en C++, permite incorporarlo a otros programas en los cuales e podría guardar distintas series, usuarios, historiales de los resultados, control por voz etc.

Figura 6.2: Escena en centro ocupacional

El objetivo principal es mantener y/o mejorar habilidades cognitivas como: memoria a corto plazo, atención, concentración, percepción viso-espacial, etc.

Al tratarse de un algoritmo preparado para cualquier tipo de objeto que cumpla una forma geométrica admitida por el programa, permite que el terapeuta use cualquier objeto a su alcance, objetos de los cuales el paciente ya puede estar familiarizado, vasos, pelotas, cajas de distinto tipo, botellas, etc.

La aplicación puede ir dirigida a una población de cualquier edad que hayan sufrido un traumatismo craneoencefálico, un accidente cerebro vascular o algún tipo de demencia como demencia mixta, alzheimer, demencia senil, demencia de pick, etc.

Considero importante el uso de las nuevas tecnologías en la terapia, ya que la esperanza de vida cada vez es mayor y es interesante aportar soluciones para mejorar la calidad de vida de las personas con problemas.

Bibliografía

1. Laboratorio de robótica Universidad Carlos III Madrid. Imágenes e información. Disponible en Web: <http://roboticslab.uc3m.es/roboticslab/index.php>

2. TedCas. Imágenes e información. Disponible en Web: <http://www.tedcas.com/index.php/es/>

3. Bilibot. Imágenes e información. Disponible en Web: <http://www.bilibot.com/>

4. Willow Garage. Imágenes e información. Disponible en Web: <http://www.willowgarage.com>

5. SCARAMUZZA, Davide. *1-Point-RANSAC Structure from Motion for Vehicle MountedCameras by Exploiting Non-holonomicConstraints*.Abril 2011.

6. PAPAZOV, Chavdar*. An Efficient RANSAC for 3D Object Recognition in Noisy and Occlude Scenes*. Múnich, Alemania: Noviembre 2010.

7. JOYANES A., Luis*. C++ Manual de bolsillo*.Madrid, España: Mc Graw Hill.1993.

8. HARVEY M.,Deitel. *C++ How to program*.Prentice Hall. 2007

9. PassThroughPCL. Documentación y tutoriales. Disponible en Web:<http://docs.pointclouds.org/trunk/classpcl_1_1_pass_through.html>

10. SACSegmentationFromNormalsPCL. Documentación y tutoriales. Disponible en Web:<http://docs.pointclouds.org/trunk/classpcl_1_1_s_a_c_segmentation_from_normals.html>

11. RandomSampleConsensusmodel PCL. Documentación y tutoriales. Disponible en Web:<http://pointclouds.org/documentation/tutorials/random_sample_consensus.php#random-sample-consensus>

12. MARTIN A., Fischeler y ROBERT C., Bolles. *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*.Nueva York, EEUU: Junio 1981. Volumen 24. Tema 6. 381–395

13. KdTree PCL. Documentación y tutoriales. Disponible en Web:<http://pointclouds.org/documentation/tutorials/kdtree_search.php#kdtree-search>

14. MORENO S., Francisco. *Clasificadores eficaces basados en algoritmos rápidos de búsqueda del vecino más cercano*. Alicante, España: Febrero 2004.

15. J.L. Bentley.*Multidimensional binary search trees used for associative searching*. 1975.

16. J.H. Friedman, J.L. Bentley, y R.A. Finkel. *An algorithm for finding best matches in logarithmic expected time*. 1977.